A METHOD AND APPARATUS FOR AUTOMATIC FOCUSING OF A CONFOCAL LASER MICROSCOPE

CROSS-REFERENCE TO ATTACHED APPENDICES

Appendices A-E (2 sheets of 158 frames) that are attached herewith, are parts of the present disclosure and are incorporated herein by reference in their entirety. Appendices A and B are a listing of routines that are used in microprocessors of coarse and fine Z controllers in two alternative embodiments of a microscope system. Appendix C is a listing of routines that are used in a programmer to program a Programmable Logic Device (PLD). Appendix D is a listing of components used in one embodiment of a microscope system. Appendix E is a listing of routines that implement computations for various auto-focus operations.

Appendix F is a listing of host workstation software that sets up various parameters for use in the area scan method. Appendix G, entitled "Ultrapointe Model 1000 Laser Imaging System Users' Manual" attached herewith is part of present disclosure and is incorporated herein by reference in its entirety.

NOTICE OF COPYRIGHT RIGHTS

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates generally to a method and apparatus for automatic focusing. More particularly, the present invention relates to automatic focusing of a confocal microscope.

BACKGROUND OF THE INVENTION

Certain prior art auto-focus microscopes generate an electronic focus signal, that is related to the distance of the target from an objective lens of the microscope, to determine the position at which the target is in focus (henceforth "focus position").

In other auto-focus microscopes, a white light camera is used to detect a maximum contrast in the camera picture to indicate when the target is in focus. Measuring the picture contrast requires a large amount of computer processing. This method is therefore slow and complicated. Additionally, this method fails if the target has no contrast (i.e., if the target is flat and has a uniform color, or is mirror-like).

The above method can fail depending on the amount of noise in the electronic focus signal, such as the noise introduced by vibration of the target during relative movement between the target and the microscope. Such noise can result in detection of a false focus position when a microprocessor uses a low power objective lens. A low power objective lens is a lens that has a low power, such as 1.5X or 5X, as opposed to a high power objective lens that has a high power, such as 100X or 200X.

Use of a low power objective lens can result in a large depth of focus, for example, greater than half the range of motion of the target with respect to the microscope. A large depth of focus combined with noise can result in an electronic focus signal that does not have a single distinct well defined peak that is essential for proper operation of certain conventional microscopes. In such a case, a microscope can use a false peak to move the target to a position at a distance from the focus position, thereby requiring manual focusing.

In certain conventional microscopes, an electronic focus signal that is used for automatic focusing is generated from an illumination spot held stationary on the target. The illumination spot could be broad band white light or monochromatic laser light. Local height variations in the surface of the target can cause the focused condition for the entire filed of view to be less than optimal if the spot at which focusing is performed is significantly higher or lower than the elevation of the remainder of the target's surface in the field of view. Therefore in targets, such as semiconductor substrates, that consist of highly regular orthogonal lines, for example, row and column bit lines of a DRAM, the microscope can produce different results depending on whether a portion of a DRAM line or a portion of the substrate between two DRAM lines lies within the spot. If the spot is on a portion of the substrate in a trench, the microscope focuses on the bottom of the trench. If the spot is on the top of a levee, the microscope focuses on the top of the levee.

However, it is desirable that the microscope focus on the same layer all the time, for example, on the top layer, regardless of what layer happened to be within the spot, and regardless of the materials on the top layer. In the multi-layer structure of a substrate, the different layers can be of different materials with different reflectivities. In a typical wafer, the top layer is a transparent passivation layer, and a predetermined layer that is typically desired is under the top layer. There is a need to focus on such a predetermined layer even if the predetermined layer does not have the highest reflectivity.

Speed, accuracy and repeatability are additional desirable characteristics for focusing a microscope. The time required to automatically focus a microscope determines the number of targets that can be viewed in a given time period, or, equivalently, determines the amount of time required to view a given number of targets, and thereby determines the cost associated with viewing each sample.

Certain prior art microscopes utilize auto-focus optics that are separate from the imaging optics of the microscope. In such microscopes, any "drift" between the auto-focus optics and the imaging optics results in loss of auto-focusing accuracy and repeatability.

It is therefore desirable to have a microscope that can automatically focus on a predetermined layer of a target quickly, accurately and repeatably.

SUMMARY OF THE INVENTION

In accordance with the present invention a confocal microscope system (henceforth "microscope system") uses a median point method for a coarse autofocus operation and an area scan method for a fine autofocus operation to provide accurate, repeatable and high-speed automatic focusing on any predetermined layer of a target. During an auto-focus operation, such as a coarse auto-focus operation or a fine auto-focus operation, the microscope system moves a target with respective to an objective lens, while an electronic focus signal is measured. In a coarse auto-focus operation, the value of the electronic focus signal is recorded periodically at large distances between a target's elevation, as compared to smaller distances used in a fine auto-focus operation.

The electronic focus signal generated by the microscope system is a novel signal that has a magnitude proportional to the amount of light reflected by the target. The electronic focus signal reaches a maximum (sometimes referred to as "peak") when the target is in focus. The electronic focus signal is used to control an auto-focus operation in which the target is moved to its focus position (an elevation at which the target is in focus) with respective to the objective lens of a confocal microscope. Such an electronic focus signal provides a very precise indication of the focus position of the target. In one embodiment, the electronic focus signal is also used to generate an image of the target. Since the same signal (electronic focus signal) is used for both focusing and imaging, a good image signal is obtained after the auto-focus operation. The narrow peak also allows the microscope system to discriminate between semi-transparent layers of the target.

In a coarse auto-focus operation, when using an objective lens having low power (e.g. 1.5x or 5x), the microscope system moves the target in a start-up move to a predetermined starting position so that the direction of the focus position is in a first predetermined direction (e.g. positive Z-axis direction) with respect to the starting position. Then the microscope system moves the target in a first pass (a movement during which electronic focus signal values are recorded), in the first direction through a predetermined first distance (e.g. one-third of the complete range of motion) and after completion of the target's movement calculates a first estimate of the focus position by the median point method. In the median point method, a host workstation calculates the sum of the recorded values and determines the target's position at which half of this sum was exceeded, to be a first estimate of the focus position. After calculating an estimate of the focus position, the microscope system moves the target to this focus position estimate.

Depending on the amount of jitter, or lack thereof in the movement of the target, additional or fewer estimates of the focus position can be calculated prior to movement of the target to the focus position. The number of estimates are kept to a minimum, to get the highest speed possible.

In one embodiment, after a first pass in a coarse auto-focus operation, the microscope system makes an optional second pass (similar to the first pass), to move the target through a predetermined second distance that includes positions on both sides of the focus position's first estimate. During the second pass, the microscope system again records the values of the electronic focus signal, and to calculates a second estimate of the focus position, again using the median point method. Although the second pass is optional, a coarse auto-focus operation for a low power objective lens that includes two passes has the advantage of accurate and repeatable focusing even in the presence of noise that typically causes conventional microscopes to malfunction. Also, such an auto-focus operation does not require a threshold and allows focus to be found quickly over a large range, but with high accuracy from the second pass. Finally, the median point method is less susceptible to stage backlash and stage motion irregularity than other conventional methods.

An auto-focus operation can be implemented with any one of three methods for generating the electronic focus signal: (a) spot method, (b) line scan method or (c) area scan method, in which a microscope system measures the value of an electronic focus signal for each elevation of the target (a) as the illumination spot is held stationary, (b) as the illumination spot is scanned along a line, or (c) as the illumination spot is scanned in an area in the field of view respectively. In a line scan method and an area scan method, the electronic focus signal can be measured at discrete spots or alternatively measured continuously over an infinite number of spots. In each of these three methods, the microscope system moves the target through several elevations, and determines the elevation that generates the peak (e.g. largest value) of the electronic focus signal.

Automatic focusing can be based on a line scan, so that the electronic focus signal is generated for several spots along a line segment on the target and the microscope automatically focuses on the brightest feature of the target that is in that line segment.

Automatic focusing can be based on an area scan, using an area peak detector (sometimes referred to as "peak detector") that generates and a microprocessor that records the largest value of the electronic focus signal as the illumination spot scans an area.

The microprocessor computes an estimate of the focus position by using the recorded values of the electronic focus signal for each target elevation. In one embodiment, during a pass in a fine auto-focus operation, the microprocessor estimates the focus position to be the target elevation at which the largest recorded value (sometimes referred to as "peak value") of the electronic focus signal was generated, and the microscope system moves the target to focus on the layer ("brightest layer") that generated this largest recorded value.

A predetermined layer that the user is interested in focusing on can be reached using an offset from the brightest layer. Use of the brightest layer and the offset results in an accurate and repeatable auto-focus operation that is dependent only on the brightest layer's reflectivity and is not influenced by the shape, location, or extent of other features on a target. Such an auto-focus operation is also not dependent on prior knowledge of the reflectivity of the predetermined layer.

The invention will be more fully understood in light of the following drawings taken together with the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a simplified block diagram that illustrates a confocal microscope system according to an embodiment of the invention.

FIGs. 2A-2C show a target of a confocal microscope below the focus position, at the focus position and above the focus position, respectively, illustrating, at each position, the pattern of light reflected from the target.

FIG. 3A illustrates the relation between the magnitude of an idealized electronic focus signal of a confocal microscope and the position of a target with respect to an objective lens as the target is moved along the Z-axis.

FIG. 3B illustrates an idealized electronic focus signal after the auto-focus system has been initialized.

FIG. 3C illustrates an actual electronic focus signal for a microscope system that uses a low power objective lens.

FIG. 4 is a block diagram of one embodiment of a Z-axis controller used to control a fine Z-stage and to provide feedback to a coarse Z-stage.

FIGs. 5A-5E are schematic diagrams of the embodiment of the Z-axis controller of FIG. 4.

FIG. 6 is a graphic representation of three passes performed during a coarse auto-focus operation according to the invention.

FIG. 7 is a graphic representation illustrating one embodiment of a fine auto-focus operation.

FIG. 8A is a graphic representation illustrating another embodiment of a fine auto-focus operation.

FIGs. 8B and 8C illustrate the movement of a target along the Z axis of FIG. 8A with respect to time in two alternative embodiments.

FIG. 9 is an isometric view of a fine Z-stage,

FIG. 10 is a cross sectional view of the fine Z-stage of FIG. 9.

FIG. 11 is an exploded isometric view of a piezoelectric element, sensor and bottom plate of the fine Z-stage of FIG. 9.

FIG. 12 is a block diagram of another embodiment of a Z-axis controller used to control a fine Z-stage and to provide feedback to a coarse Z-stage.

FIGs. 13A, 13B and 13C are timing diagrams illustrating various signals in one embodiment of a confocal microscope system.

FIGs. 14-21 are schematic diagrams of the embodiment of the Z-axis controller of FIG. 12.

DETAILED DESCRIPTION

FIG. 1 is a simplified block diagram of a confocal microscope system 100 described briefly below and described in more detail in commonly owned, copending U.S. Patent Application, Serial No. 08/080,014, entitled "Laser Imaging System for Inspection and Analysis of Sub-Micron Particles", filed by Bruce W. Worster et al, on June 17, 1993, that is incorporated herein by reference in its entirety.

Laser 102 (FIG. 1) generates a laser beam 123I that is transmitted through a polarizing beam splitter 104, a converging lens 130, a spatial filter 131 (such as a pinhole), a collimator lens 132, a quarter waveplate 133, reflected from an X-mirror 106 and a Y-mirror 108, and transmitted through an objective lens 110 to the surface of a target 112. In one embodiment, laser 102 is a conventional argon-ion laser, however, other types of lasers may be utilized in alternate embodiments. Target 112 is an object, such as a semiconductor wafer, that is to be viewed using microscope system 100. X-mirror 106 and Y-mirror 108 are each rotatable about an axis such that the illumination spot created by incident laser beam 123I can be moved along an X-axis and a Y-axis, respectively, of target 112. Laser 102, beam splitter 104, converging lens 130, spatial filter 131, collimator lens 132, quarter waveplate 133, X-mirror 106, Y-mirror 108 and objective lens 110 are each conventional structures that are well known by those skilled in the art of confocal microscopes.

Laser beam 123I reflects from the surface of target 112 in a pattern (illustrated in FIGs. 2A-2C) that is dependent upon the distance of objective lens 110 from target 112. FIGs. 2A-2C show target 112 below focus position 203, at focus position 203 and above focus position 203, respectively. When target 112 is positioned below or above focus position 203, respectively, a small percentage of light 123R from incident laser beam 123I that is originally transmitted through objective lens 110 is reflected back through objective lens 110 in a coherent manner. The amount of reflected light 123R is illustrated by the graph of the electronic focus signal as a function of position in FIG. 3A. There is a gradual transition in the amount of reflected light (depending on the target's position) between FIGs. 2A-2C. As is well known in microscope engineering art, the amount of reflected light also depends on, for example, reflectivity of target, angle of target and angle of the chuck that supports the target.

Microscope system 100 includes a spatial filter 131 with a pin hole that permits passage of incident laser beam 123I and some portion of reflected light 123R, the portion depending on the position of target 112. Spatial filter 131 blocks off the rest of reflected light 123R as illustrated in FIGs. 2A and 2C. However, when target 112 is positioned at focus position 203 (FIG. 2B), a maximum amount of light from incident laser beam 123I is reflected (light 123R) and transmitted back through objective lens 110 and the pinhole along a conjugate path of microscope system 100.

Reflected light 123R passes through objective lens 110, is reflected by Y-mirror 108 (FIG. 1), X-mirror 106 and reflects off beam splitter 104 to a photodetector 114. Photodetector 114 is a device, such as a photo-multiplier tube (PMT) or photo-diode, that generates an electronic focus signal 115 that is an analog signal that has a magnitude (voltage in one embodiment) proportional to the intensity of reflected light 123R as measured by photodetector 114. The photodetector's gain is adjustable to accommodate different laser power, laser wavelength and target reflectivity. As is well known in microscope engineering, when the target is in focus, the value of the detected electronic focus signal is proportional to: PMT's GAIN * TARGET'S REFLECTIVITY * LASER'S POWER. The PMT's gain is adjusted empirically, as discussed below.

Electronic focus signal 115 (FIG. 1) is provided to host workstation 116 and to Z-axis controller 118. Z-axis controller 118 is directly coupled to fine Z-stage 120 and is indirectly coupled to coarse Z-stage 122 through host workstation 116 and coarse Z-axis controller 117. Coarse Z-stage 122 uses a motor, such as a stepper motor, to move target 112 relative to objective lens 110 through a relatively large range of motion (e.g. a large percentage of total possible movement) along the Z-axis, in an operation referred to as "coarse auto-focus operation". In one embodiment, coarse Z-stage 122 moves target 112 through 1000 microns in a start-up move of a coarse auto-focus operation to a starting position 516 (FIG. 6) from an initial position of target prior to the auto-focus operation (the total possible movement being approximately 6000 μm microns).

In one embodiment of the present invention, coarse Z-axis controller 117 is a conventional stepper motor controller available as part number 310MX3 from New England Affiliated Technology (NEAT) of 620 Essex St., Lawrence, MA 01841. Coarse Z-stage 122 can be driven by a conventional stepper motor (not shown) such as the Vexta C5858-9012 available from Oriental Motor of 16-17 Veno 6-Chome Taito-Ku, Tokyo, Japan. In this embodiment, coarse Z-stage 122 is Part # 1930237 available from NEAT (above).

As explained in more detail below, fine Z-stage 120 uses a piezoelectrically driven element to move target 112 through a relatively small range of motion (e.g. a small percentage of total possible movement) along the Z-axis, in an operation referred to as "fine auto-focus operation." In one embodiment, fine Z-stage 120 moves target 112 through 50 microns in a fine auto-focus operation (the total possible movement being approximately 6000 μm microns). Although the invention is described as having a movable target 112 and a stationary objective lens 110, target 112 can be held stationary while objective lens 110 is moved.

FIG. 3A is an idealized graph of the magnitude (sometimes referred to as "strength" or "intensity") of an electronic focus signal 115A as a function of the position, e.g. elevation of target 112 along the Z-axis. Electronic focus signal 115A has a magnitude that is theoretically proportional to a sinc-squared function $((\sin(x)/x)^2)$, having a full width half max measurement 305A that varies based on the numerical aperture of objective lens 110 and the wavelength of laser beam 123. The full width half max measurement 305A is the width, along the Z-axis, between two points 308A and 308B at which the magnitude of electronic focus signal 115A is at half of its maximum magnitude 301A at focus position 203. As illustrated in FIG. 3A, electronic focus signal 115A has a value of (A1+A2)/2 for the full width half max measurement 305A, wherein A1 is the background value (described below) and A2 is the maximum value for focus position 203. The relationship of the numerical aperture and the wavelength to the magnitude of the electronic focus signal is well known to a person skilled in the art of confocal microscopes.

In FIG. 3A, focus position 203 at the center of main lobe 307A is a distinct position, as shown by a sharp, well defined peak 301A in the magnitude of electronic focus signal 115A. Electronic focus signal 115A also exhibits side lobes such as lobes 306a-306b. Depth of focus (sometimes referred to as "width of focus") 302A spans a range along the Z-axis in which the magnitude of electronic focus signal 115 is greater than a background value 303. Background value 303 is a small value of electronic focus signal 115A that is non-zero and results from leakage currents and a small amount of background light that reaches photodetector 114.

Depth of focus 302A becomes smaller as the numerical aperture of objective lens 110 increases or as the wavelength of laser beam 123 decreases, as is well known. In the following discussion, objective lens 110 has a power of 100X and numerical aperture of 0.95, and laser beam 123 has a wavelength of 488 nm in one embodiment. In this embodiment, electronic focus signal 115A has a full width half max measurement 305A of approximately 0.5 microns.

Microscope system 100 automatically moves target 112 to an estimate of focus position 203 that results from one or more auto-focus operations as described below. To initialize microscope system 100 for a coarse auto-focus operation, the gain of photodetector 114 is increased to an empirically predetermined autofocus gain. Also, an empirically predetermined zero position offset 310 is applied to electronic focus signal 115 such that background value 303 and side lobes 306a-306b are effectively eliminated from electronic focus signal 115A. FIG. 3B illustrates electronic focus signal 115A after microscope system 100 has been initialized.

The auto-focus gain of photodetector 114 (FIG. 1) and zero position offset 310 (FIG. 3B) are empirically determined by performing a series of auto focus operations using different targets 112 to cover as wide a range of targets as typically used by the user (e.g. an aluminum target and a semiconductor target). Zero position offset 310 is selected to ensure that background value 303 and side lobes 306a-306b are below zero position 320. A small positive value greater than zero position 320 is then selected for use as threshold value 304. Once threshold value 304 has been selected, the threshold value 304 can be used for subsequent auto-focus operations on a wide variety of targets.

When the magnitude of electronic focus signal 115A exceeds a predetermined threshold value 304, a near-focused condition is said to exist. During a near-focused condition, target 112 is relatively close to focus position 203 (illustrated by positions 308a and 308b of FIG. 3A). The gain of photodetector 114 and the zero position offset 310 are then maintained during the coarse auto-focus operation.

Certain photodetectors 114, particularly photo multiplier tubes (PMTs), can be damaged when exposed to high signals (laser signals having a magnitude greater than magnitude that PMTs are designed for) that result from increasing the gain of photodetector 114. Such damage is not a problem in the current invention because PMT control circuitry has an overload sensing circuit that automatically reduces PMT's gain to zero by a conventional method, before damage occurs.

FIG. 3C illustrates variations in magnitude of an actual electronic focus signal 115C. Magnitude of electronic focus signal 115C is irregular due to noise caused by, for example, vibration between target 112 and objective lens 110 during movement of target 112. In FIG. 3C, objective lens 110 has low power (for example magnification of 1.5X or 5X). The low power of objective lens 110 results in a large depth of focus, such as distance 305C (e.g. 30-100 microns) between elevations 335A and 335B for the full width half max measurement. The large depth of focus and the irregular waveform of electronic focus signal 115C can result in a false focus position, such as one of position 333A or 333B, if peak detection is used to estimate focus position 203.

Use of a threshold, as described above in reference to FIGs. 3A and 3B, when electronic focus signal 115C is generated by a low power objective lens has the disadvantage that a small change in threshold, (e.g. from 0.5 to 0.6) results in a large change in the target's elevation along the Z axis (e.g. from points 335A and 335B to points 336A and 336B) at which the threshold is exceeded. In such a case, focus position

203 of idealized main lobe 307 is estimated in one embodiment without using a threshold, by summing up the magnitude of electronic focus signal 115C at several elevations of target 112 in the range of motion along the Z-axis, in a median point method that is described below.

FIG. 4 is a block diagram of Z-axis controller 118 that controls fine Z-stage 120 and also provides feedback used to control coarse Z-stage 122. Operation of some structures in FIG. 4 is described below, while operation of the rest is described in reference to FIGs. 5A-5E and 6-8.

Within Z-axis controller 118 (FIG. 4), electronic focus signal 115 is transmitted to a first input terminal of comparator 401 and to an input terminal of an analog to digital converter (ADC) 407. ADC 407 measures the magnitude of electronic focus signal 115 when commanded by microprocessor 403, for example eighty (80) times a second. A digital output signal (on 8 lines in one embodiment) of ADC 407 drives microprocessor 403. Comparator 401 sets a flip-flop 402 when the magnitude of electronic focus signal 115 exceeds a threshold signal at the comparator's second input terminal that is driven by a digital to analog converter (DAC) 404. DAC 404 in turn receives a threshold value from microprocessor 403.

The output terminal of comparator 401 (FIG. 4) is coupled to the set terminal of latching flip flop 402. The Q output terminal of flip-flop 402 is coupled to an input terminal of status register 405.

An output terminal of control register 406 is coupled to the reset terminal of flip flop 402. Control register 406 resets flip-flop 402 on command from microprocessor 403.

Microprocessor 403 has terminals coupled to status register 405, control register 406, ADC 407 and host work station 116. Microprocessor 403 is also coupled to position control register 408. The output of position control register 408 is transmitted through DAC 409, summing node 410, integrator 420, and amplifier 411 to provide a control voltage signal to a piezoelectric element 1130 of fine Z-stage 120. Summing node 410 also receives a feedback signal (on line 435) that is linearly proportional to the position of target 112 along the Z-axis i.e. distance from a proximity sensor 1135 of fine Z-stage 120. The feedback signal is used to linearize the behavior of a piezoelectric element 1130 (FIG. 11).

FIGs. 5A-5E are schematic diagrams of one embodiment of Z-axis controller 118 of FIG. 4. Similar elements in FIGs. 4 and 5A-5E are labelled with the same number. Ratings of various components illustrated in FIGs. 5A-5E are listed in microfiche appendix D (at pages 121-126) for one embodiment.

Central processing unit (CPU) 5000 (FIG. 5D) of microprocessor 403 transmits and receives information through bus transceiver 5034 to 8-bit data bus 5006. CPU 5000 is, for example, a TP-RS485 twisted pair control module, model number 55050-00, available from Echelon of 4015 Miranda Avenue, Palo Alto, CA 94304. Bus transceiver 5034 provides additional drive capability to CPU 5000. Bus transceiver 5034 is a well known device, available for example as part number 74ALS245, from Texas Instruments (TI) of 7839 Churchill Way, Dallas, TX 75251.

Address register 5036 receives addressing information from CPU 5000 through data bus 5006. This addressing information determines the register or device within Z-axis controller 118 that the microprocessor 403 accesses. Address register 5036 is available, for example, from TI (above) as part number 74ALS573.

The output of address register 5036 is provided to address decoders 5038 and 5040. Address decoders 5038 and 5040 decode the addressing information and generate signals that enable a register or device within Z-axis controller 118 that microprocessor 403 accesses. Address decoders 5038 and 5040 are available for example from TI (above) as part numbers 74ALS138 and 74LS139, respectively. Microprocessor 403 communicates with position control register 408, status register 405, control register 406, DAC 404 and ADC 407 using 8-bit data bus 5006.

Registers 5001 and 5003 (FIG. 5A) within position control register 408 receive positioning information from microprocessor 403 on data bus 5006. Registers 5001-5004 are well known in the art of designing microprocessor based systems. Registers 5001 and 5003 are available for example from TI (above) as part number 74ALS574. Registers 5002 and 5004 are available for example from TI (above) as part number 74ALS273. Eight bit words on data bus 5006 are transmitted through registers 5001-5004 of position control register 408 to provide a 12-bit input to 12-bit DAC unit 5008 of DAC 409.

DAC unit 5008 is a conventional digital to analog converter (DAC), known in the electronics art, and available for example from Analog Devices (AD) of 1 Technology Way, Norwood, MA 02062, as part number AD7541. The remaining ancillary elements of DAC 409 including operational amplifier 5009 and the illustrated resistors, capacitors and diodes are conventional elements commonly seen in conventional circuits using DAC 409, as is well known to a person skilled in the art of electronic engineering. Operational amplifier 5009 is available for example from AD (above) as part number OP-177E. DAC 409 provides an analog output signal on lead 5010.

As shown in FIG. 5B, lead 5010 is connected to one input terminal of operational amplifier 5012 of summing node 410. Operational amplifier 5012 is available for example as part number AD712 from AD (above). The signal on the other input terminal operational amplifier 5012 is derived from the position feedback signal (on line 435 in FIG. 4) of sensor 1135 (FIG. 11) in fine Z-stage 120. Sensor 1135 provides an input signal to operational amplifier 5018. Operational amplifier 5018 is available for example from AD (above) as part number AD712.

Various illustrated resistors and capacitors, such as R7, C8, R8 that are coupled to operational amplifier 5018 create a conventional buffer. Components rated as DNL (Do Not Load) in FIGs. 5A-5E and FIGs. 14-21 are not used (i.e. not part of the circuit). The output signal of operational amplifier 5018 is provided to the other input terminal of operational amplifier 5012. The output terminal of summing node 410 is coupled to the input terminal of integrator 420. Integrator 420 includes an operational amplifier such as part number AD712, available from AD (above). The associated resistors, capacitors and diodes such as R19, C20, R20, R21, C25, D5 of integrator 420 are known to one skilled in the art of designing electronics.

The output signal of integrator 420 is provided to notch filter 5013, which includes two resistors (e.g. R24, R23) and two capacitors (e.g. C22, C27). The output signal of notch filter 5013 is provided to operational amplifier 5014, that is available, for example, from AD (above) as part number AD712. The output signal of operational amplifier 5014 is provided to the input terminal of amplifier 411.

Amplifier 411 is a conventional amplifier that includes an operational amplifier available from for example Apex Microtechnology of 5980 N. Shannon Rd., Tucson, AZ 85741 as part number PA85. The illustrated diodes, resistors and capacitors such as R22, D3, D4, D8, C34, R14, C14, R5, R6, C7, R2, C1, R4 of amplifier 411 are all known in the electronics art. The output signal of amplifier 411 is provided to piezoelectric element 1130 within fine Z-stage 120.

As shown in FIG. 5C, electronic focus signal 115 is provided to ADC 407. Electronic focus signal 115 is routed through multiplexer 5022 to operational amplifier 5020. Multiplexer 5022 is a conventional part available, for example from Siliconix of 2201 Laurelwood Road, Santa Clara, CA 95058 as part number DG 211. Operational amplifier 5020 is available from for example AD (above) as part number AD843. Operational amplifier 5020 buffers electronic focus signal 115. The output of operational amplifier 5020 is provided to an input of ADC unit 5021. ADC unit 5021 is a conventional part available for example as part number AD7575 from AD (above). The other devices C17, C18, D2, R26, C23, C51, C52, U27 coupled to ADC unit 5021, as illustrated in FIG. 5C, are known in the art of electronics. In response to electronic focus signal 115, ADC unit 5021 supplies an 8-bit digital signal representative of electronic focus signal 115. The 8-bit digital signal of ADC unit 5021 is provided to microprocessor 403 on data bus 5006.

FIG. 5C also illustrates flip flop 402. Flip flop 402 is programmed as one of the devices present within programmable logic device (PLD) 5023. PLD 5023 is available for example from Lattice Semiconductor of 5555 NE Moore Ct., Hillsboro, OR 97124, as part number GAL20RA10. The input signals to PLD 5023 include: a set input signal from comparator 401 and a reset input signal from control register 406. PLD 5023 processes these input signals and generates a Q output signal representing the output signal at Q output terminal of flip flop 402. This Q output signal is provided to status register 405. (PLD 5023 also has input signals and output signals unrelated to auto-focus operations.) One embodiment of instructions to a programmer for programming PLD 5023 are listed microfiche appendix C that is incorporated herein by reference in its entirety. Instructions in microfiche appendix C can instruct programmer model no. BP-1200 available from BP Microsystems, Inc. of 1000 N. Post Oak Road, Houston, TX 77055.

FIG. 5D illustrates status register 405. Status register 405 is a conventional register available for example as part number 74ALS541 from TI (above). As previously discussed, status register 405 receives the Q output signal of flip flop 402 from PLD 5023 (status register 405 also receives other information unrelated to auto-focus operations). The 8-bit output signal of status register 405 is provided to data bus 5006 such that microprocessor 403 can detect when flip flop 402 is set.

Control register 406 (FIG. 5D) receives an 8-bit input signal from microprocessor 403 on data bus 5006. Control register 406 is available for example from TI (above) as part number 74ALS273. An output terminal of control register 406 is coupled to PLD 5023, such that a signal from control register 406 can reset flip flop 402.

DAC 404 also receives an 8-bit input signal from microprocessor 403 on data bus 5006. This 8-bit input signal is transmitted through register 5007 (available for example from TI as part number 74ALS574) to conventional DAC unit 5011 (available for example from National Semiconductor as part number DAC0808). DAC unit 5011 converts the incoming 8-bit signal into an analog output signal. This analog output signal is provided to an input terminal of operational amplifier

5017 (available for example from AD as part number AD712). The output signal of operational amplifier 5017 is provided to an input terminal of comparator 401. Electronic focus signal 115 is provided to the other input terminal of comparator 401. Comparator 401 includes comparator unit 5019, available for example from National Semiconductor of 2900 Semiconductor Drive, Santa Clara, CA 95062, as part number LM311. The output signal of comparator 401 is provided to flip flop 402.

FIG. 5E illustrates conventional structures used in the power supply connections 5041, 5042 that supply power to fine Z-stage controller 118. Analog/digital grounding structure 5043 is used to connect analog and digital grounds on fine Z-stage controller 118.

In one embodiment of the present invention, a coarse auto-focus operation is performed as follows. Host work station 116 (FIG. 1), through coarse Z-stage controller 117, instructs coarse Z-stage 122 to move target 112 in a start-up move (not shown) downward through a startup predetermined distance (e.g. 1000 microns) along the Z-axis from the current position of target 112. If there is less distance from the start-up predetermined distance between the current position and a lowermost position (not shown) along the Z-axis, then target 112 is moved to the lowermost position along the Z-axis. The lowermost position is that position along the Z-axis below which target 112 cannot be moved by any mechanism in microscope system 100. This startup pass ensures that target 112 is positioned at a starting position 516 below focus position 203. For example, as all focus positions are within a limited distance (≈ 200 μm in one embodiment) of an upper travel limit, a position e.g. 1000 μm below any initial position (not shown) guarantees that starting position 516 is below focus position 203. Travel limits are two positions of a target that are farthest from each other, such as a lower most position (not shown) and first safe operating position 520 of FIG. 6. If the first time a target is loaded, coarse Z stage 122 is at its lowermost position, there is no startup pass. A startup pass also ensures that target 112 is positioned at least a minimum distance below focus position 203, thereby allowing coarse Z-stage 122 to achieve consistent start-up characteristics (i.e., velocity, acceleration, etc.) before target 112 encounters a focused condition.

Host work station 116 instructs microprocessor 403 to send a digital signal to DAC 404 such that the output signal of DAC 404 has a voltage level corresponding to threshold value 304 that was determined during initialization of microscope system 100. In one embodiment, the input signal to DAC 404 is a fixed digital input signal having a value of 24 (out of a range of values between 0 and 255). In response to another signal generated by host workstation 116, control register 406 transmits a reset signal to reset flip-flop 402 to its initial state (e.g., a logic "0").

FIG. 6 is a graphic representation of three coarse passes 513, 514 and 515 that are performed during a coarse auto-focus operation according to one embodiment of the invention. A number of coarse passes other than three can be used in other embodiments. The vertical axis in FIG. 6 illustrates the position (e.g. elevation) of target 112 (FIG. 1) along the Z-axis. The horizontal axis in FIG. 6 illustrates the magnitude of the electronic focus signal 115.

To perform a first coarse pass, host work station 116 (FIG. 1) instructs coarse Z-stage controller 117 to move (illustrated by arrow 512) coarse Z-stage 122, and thereby target 112, from starting position 516 in positive Z direction 533 through a first safe operating distance 512 to a first safe operating position 520. If no focus condition is detected during a first coarse pass, target 112 is moved to first safe operating position 520. If a focus position is detected, the movement of target 112 is illustrated by the three coarse passes, 513, 514 and 515.

First safe operating distance 512 is selected so that there is no chance that target 112 will contact objective lens 110 that is located at position 517. Any amount of clearance between first safe operating position 520 and position 517 can be selected. In one embodiment, the focus position of each objective lens is measured and travel limits are programmed (e.g. soft coded into a software configuration table) so that no contact occurs between target 112 and objective lens 110. All entries in a configuration table are defined by a user during installation of microscope system 100. In particular, travel limits for movement of the target are set by manual focusing on a target. In one embodiment, first safe operating position 520 is empirically set at 50 microns above focus position 203 (i.e., 50 microns above the focal point of objective lens 110) based on various parameters such as thickness of target and the working distance (e.g. 270 microns) as described below. In one embodiment, first safe operating distance 512, is selected from the range of 1000-6500 microns, depending on the initial position of target 112, i.e., before target 112 is moved to starting position 516.

The working distance of objective lens 110 is the distance from the objective len's position 517 (FIG. 6) to focus position 203. In various embodiments, the working distance varies from 270 microns to several millimeters, depending on the numerical aperture of objective lens 110.

Safe operating position 520 depends on many factors, such as flatness of the target, flatness of the XY stage on which the target is supported, position of fine Z stage 120, thickness of the target and repeatability of a "home" position of coarse Z stage 122. Safe operating position 520 is empirically selected to be above a typical focus position 203 for a typical target, but below the position at which the target touches objective lens 110. Safe operating position 520 is selected to be sufficiently away from focus position 203 to ensure enough travel to obtain a well defined peak in the magnitude of electronic focus signal 115.

In a threshold method, during first coarse pass 513, comparator 401 continuously compares incoming electronic focus signal 115 with threshold value 304 received from DAC 404. Because target 112 is initially out of focus, electronic focus signal 115 is initially less than threshold value 304. Under these conditions, the output signal of comparator 401 has a positive voltage. As target 112 approaches focus position 203, the magnitude of electronic focus signal 115 increases. When the magnitude of electronic focus signal 115 exceeds threshold value 304, the output signal of comparator 401 transitions to a negative voltage, thereby setting flip flop 402.

Once flip flop 402 (FIG. 4) is set, the voltage at the Q output terminal transitions to a logic high state in this embodiment. Such a logic high state at the Q output is transmitted to status register 405, causing a bit within status register 405 to change value. Microprocessor 403, which continuously monitors status register 405, thereby detects that flip flop 402 has latched. Upon detecting this latched condition, microprocessor 403 signals host work station 116. In response, host work station 116 instructs coarse Z-axis controller 117 to stop coarse Z-stage 122, and thereby stop movement of target 112.

Because electronic focus signal 115 is an analog signal and flip flop 402 latches when the magnitude of electronic focus signal 115 exceeds threshold value 304, the possibility of missing a focused condition is eliminated if the peak magnitude is greater than the threshold value. There are no discrete sampling periods during which the focused condition may be missed. Thus, flip flop 402 latches even for a focus signal 115 having a narrow depth of focus 302, as illustrated in FIG. 3A. Consequently, target 112 can be moved upward at a much faster velocity than was possible with auto-focus microscopes of the prior art.

In one embodiment, the average velocity of target 112 during first coarse pass 513 is dependent on first safe operating distance 512. Target 112 is moved at an average velocity that enables target 112 to move through first safe operating distance 512 in approximately one second. However, the maximum average velocity is approximately 3000 microns per second in this embodiment. This allows first coarse pass 513 to be completed in approximately 1 second.

If a focus condition is not detected before target 112 completes movement through first safe operating distance 512, host work station 116 instructs coarse Z-stage controller 117 to stop target 112 when first safe operating position 520 is reached. A focus position can remain undetected depending on the parameters used to detect focus position 203. For example, when a threshold value is larger than the peak magnitude of electronic focus signal 115, the output of comparator 401 does not transition to a negative voltage even as target 112 passes through focus position 203.

Because of the high velocity (e.g. 3000 microns per second) at which target 112 is moved during first coarse pass 513 and the significant amount of time (e.g. 20-60 milliseconds) required to stop the movement of target 112 after a focused condition is detected, target 112 moves ("target overshoot") to first stopping position 505 (FIG. 6) above focus position 203 at the time that target 112 comes to rest. This target overshoot is illustrated in FIG. 6, which shows that a focused condition is detected at first trip position 504, and that target 112 comes to rest at first stopping position 505. Thus, it is necessary to perform one or more additional passes to position target 112 closer to focus position 203.

The distance required to stop target 112 during first coarse pass 513 is dependent upon threshold value 304 (because a lower threshold value 304 causes flip flop 402 to latch sooner), the system gain (because a higher system gain allows for earlier detection of a focused condition), the time required for microprocessor 403 to detect that flip flop 402 has latched, the time required for microprocessor 403 to communicate this information to host work station 116, the time required for host work station 116 to issue a command to stop coarse Z-stage 122, and the velocity of coarse Z-stage 122 at first trip position 504. Such dependencies are well known to a person skilled in engineering.

Host work station 116, through microprocessor 403, then instructs control register 406 to transmit another reset signal to clear flip flop 402 (FIG. 4) before second coarse pass 514 is begun. This reset signal allows flip flop 402 to detect the next time electronic focus signal 115 exceeds threshold value 304.

The steps performed during second coarse pass 514 are similar or identical to those of first coarse pass 513. Host work station 116 instructs coarse Z-stage 122 (through coarse Z-stage controller 117) to move target 112 in negative Z direction 532 at a velocity that is slower than the velocity of target 112 during first coarse pass 513. This motion continues until a near-focused condition is detected at second trip position 506, causing flip flop 402 to latch, or until target 112 moves through a predetermined second safe operating distance 507 to second safe operating position 522. This second safe operating distance 507 is selected to assure that target 112 travels through focus position 203, based on the width of electronic focus signal 115 and the range of anticipated overshoot of target 112 during first coarse pass 513. Second operating distance 507 is selected greater than the overshoot from first coarse pass 513, plus the width of electronic focus signal 115.

In one embodiment, second safe operating distance 507 is 1200 microns, unless target 112 was stopped less than 50 microns from first safe operating position 520. If target 112 was stopped less than 50 microns from first safe operating position 520, the second safe operating distance 507 is smaller, e.g. 200 microns in this embodiment. The latter second safe operating distance is smaller because target 112 is decelerating near the end of first safe operating distance 512 so that it is easier to stop target 112, thereby decreasing the amount of target overshoot so that target 112 is stopped closer to focus position 203 than would otherwise be the case. Because target 112 is closer to focus position 203, a smaller second safe operating distance 507 can be used.

In one embodiment, it takes approximately 1 second for target 112 to pass through the entire second safe operating distance 507. The average velocity of target 112 during second coarse pass 514 therefore is approximately 200 microns per second or 1200 microns per second, depending upon second safe operating distance 507.

While traversing second safe operating distance 507, target 112 encounters a focused condition at second trip position 506. When electronic focus signal 115 exceeds threshold value 304, flip flop 402 latches and target 112 is stopped in the same manner as in first coarse pass 513. Target overshoot results in target 112 coming to rest below first trip position 504. In FIG. 6, target 112 stops at second stopping position 508 and a microscope system 100 estimates the location of focus position 203 based on overshoot and width of focus.

One or more additional passes can be used to make a better estimate (than the previous estimate) of focus position 203, depending on the resolution of ADC 407.

The steps performed during third coarse pass 515 are also similar or identical to the steps described above for second coarse pass 514 and first coarse pass 513. Host work station 116, through microprocessor 403, instructs coarse Z-stage controller 117 to move target 112 in positive Z direction 533 at a velocity slower than the velocity of target 112 during second coarse pass 514. This motion continues until a near-focused condition 509 causes flip flop 402 to latch or until target 112 moves through a predetermined third safe operating distance 510 to a third safe operating position 524. Again, third safe operating distance 510 is selected to assure that target 112 travels through focus position 203 and is dependent on the amount of target overshoot associated with second coarse pass 514.

In one embodiment, the third safe operating distance 510 is 140 microns. For target 112 to take approximately one second to pass through the entire third safe operating distance 510, the average velocity of target 112 during the third coarse pass 515 is approximately 140 microns per second in one embodiment.

Prior to traversing the entire third safe operating distance 510, target 112 again encounters a near-focused condition at first trip position 504. When the electronic focus signal 115 exceeds threshold value 304, flip flop 402, having been reset, latches and target 112 is stopped in the same manner as in coarse passes 513 and 514.

Because of the relatively low velocity of coarse Z-stage 122 during third coarse pass 515, target 112 comes to rest at a third stopping position 511 that can either be above or below focus position 203. After third coarse pass 515, the third stopping position 511 of target 112 is within approximately ± 10 μm microns of focus position 203. To achieve this degree of accuracy, all of the factors contributing to target overshoot are considered to empirically determine the amount of time required to stop target 112 upon encountering a focused condition during the third coarse pass 515. Given the time required to stop target 112, the velocity of target 112 during third coarse pass 515 is selected to assure that third stopping position 511 of target 112 is within approximately ± 10 μm microns of focus position 203.

If full width half max measurement 305 of electronic focus signal 115 is sufficiently wide, the coarse auto-focus operation described above is modified. Although there is no clear boundary that determines when an electronic focus signal 115 is "sufficiently wide", an electronic focus signal 115 created with a laser beam 123 having a wavelength of 488 nm and an objective lens 110 having a numerical aperture of 0.13 or lower is considered "wide," and has full width half max measurement 305 of approximately 29 microns. Electronic focus signal 115 exhibits a wider full width half max measurement 305 as the magnification and numerical aperture of objective lens

110 decreases or as the wavelength of laser beam 123 increases. For example, use of a low power (e.g. 1.5x or 5x) objective lens results in a large depth of focus, as illustrated by electronic focus signal 115C (FIG. 3C).

When attempting a first coarse pass 513 on a "wide" electronic focus signal 115, first stopping position 505 of target 112 is relatively close to the second trip position 506. In certain cases, first stopping position 505 of target 112 is at a position where the value of electronic focus signal 115 exceeds the threshold value 304. That is, there is not enough target overshoot to guarantee that target 112 "escapes" the electronic focus signal 115. Target 112 may "escape" if as a result of first coarse pass 513, target 112 is close to second trip position 506 and is unable to obtain a high velocity before flip-flop 402 latches during second coarse pass 514. As a result, second stopping position 508 of target 112 can be well short of peak 301. This condition causes the coarse auto-focus operation described above to miss focus position 203.

In one embodiment, to perform a coarse auto-focus operation on a "wide" electronic focus signal 115 (FIG. 3C), target 112 (FIG. 1) is moved in a start-up move by 2000 microns (rather than 1000 microns described above) below the target's initial position (not shown) to starting position 516 (FIG. 6). If target 112 is within 2000 microns of the lowermost position (not shown) along the Z-axis, target 112 is moved this lowermost position. This starting position can provide target 112 with an additional distance in which to accelerate before reaching first trip position 504. This acceleration and starting position ensure that target 112 overshoots second trip position 506. The rest of the characteristics of first coarse pass 513 for wide electronic focus signal 115C are identical to those described above.

During second coarse pass 514 for wide electronic focus signal 115C, target 112 is moved downward along the Z-axis through second safe operating distance 507. The second safe operating distance 507 and the velocity of target 112 during the second coarse pass 514 is determined in the manner previously described (i.e. based on a first estimate determined by the threshold method described above). However, rather than monitoring the status of flip-flop 402 during second coarse pass 514, host work station 116 instructs microprocessor 403 to record the values of the electronic focus signal output by ADC 407 repeatedly at a predetermined interval (e.g. 80 times a second) while coarse Z-stage 122 is moving target 112 at an approximately constant velocity.

The values recorded by microprocessor 403 therefore roughly correspond to magnitude of electronic focus signal 115 at regular distances along the Z-axis for the entire predetermined second safe operating distance 507. Based on these recorded values, focus position 203 is calculated by a median point method described below. Because movement of coarse Z-stage 122 generally induces noise in the electronic focus signal 115, the values recorded by microprocessor 403 can exhibit peaks at positions other than focus position 203 (as illustrated by FIG. 3C). Microprocessor 403 therefore filters the noise to obtain a better estimate of focus position 203 in the "median point" method.

In the median point method host workstation 116 calculates the sum of the values previously recorded by microprocessor 403 and determines the elevation along the Z-axis at which half of this sum was reached during second coarse pass 514. Host workstation 116 then issues a command to move target 112 in the positive Z-direction to this position. The median point method assumes that noise in electronic focus signal 115 is due to irregularity of movement of coarse Z-stage 122 and that this irregularity occurs randomly and is equally probable on either side of the focus position 203 along the Z-axis.

In another embodiment of a microscope system 100 that uses a low power objective lens, during first coarse pass 513, instead of monitoring the status of flip-flop 402 according to the threshold method, microprocessor 403 records the values of electronic focus signal 115C (FIG. 3C) in the manner described above for second coarse pass 514 for use in the median point method. Therefore, during first coarse pass 513, coarse Z-stage 122 (FIG. 1) moves target 112 through a predetermined first safe operating distance 512 that is a safe travel limit of the movement of target 112. In this embodiment, starting position 516 is a predetermined distance of 1000 microns below the upper travel limit set by first safe operating position 520. Such a starting position 516 that is independent of the target's initial position avoids the inherent uncertainty in the location of the focus position relative to starting position if the target is moved by a predetermined distance from the target's initial position. At the end of first coarse pass 513, host work station 116 calculates a first estimate of focus position 203 by using the median point method described above.

Then host workstation 116 determines starting and stopping positions of second coarse pass 514. Starting position 531 is at a distance D (of 120 microns in one embodiment), above first estimate of focus position 203 while stopping position 522 is at the distance D after first estimated focus position 203. Distance D is chosen based on the desired precision and the accuracy of first estimate of focus position 203. If 80 values are collected during second coarse pass 514, a 240 micron range of movement yields a resolution of 4 microns.

In this embodiment, host workstation 116 uses values recorded during first coarse pass 513, to calculate (as described below) and set an auto-focus gain (e.g. gain of a PMT), so that, the intensity of the electronic focus signal during second coarse pass 514 is optimal. After the auto-focus gain is set, host workstation 116 instructs coarse Z stage 122 to move to second coarse pass starting position 531. After coarse stage 122 has moved to starting position 531, host workstation 116 instructs microprocessor 403 to start recording the values of electronic focus signal 115, and instructs coarse Z stage 122 to move to stopping position 522.

The calculation for optimal auto-focus gain depends on the specific sensor, and in one embodiment is as follows:

if "peak intensity" of the electronic focus signal is below optimal: then new auto-focus gain=old auto-focus gain + $\frac{\log(195)}{\log(\text{peak intensity})} * 9$;
if the peak intensity is above optimal, then new auto-focus gain=old auto-focus gain - (peak intensity - 195)/10,
where sensor gain is normalized to a range of 1 to 100, 1 being minimum gain, electronic focus signal intensity being normalized to the range of 0 to 225, 0 being dark. The peak intensity (e.g. 195) of the electronic focus signal is at maximum fraction (e.g. 3/4) of the maximum permitted intensity (e.g. 255), in one embodiment.

A maximum fraction (e.g. 3/4) is empirically chosen so that electronic focus signal 115 is not too high (saturation) at focus position 203 and not too low (indistinguishable from noise) at positions other than focus position 203. Other maximum fractions, such as 2/3 or 4/5 can also be chosen, and the exact fraction that is chosen is not a critical aspect of this invention. If for some reason, the peak intensity of electronic focus signal is less than a given minimum, the minimum is used in the calculation. In one embodiment, 15 is the minimum intensity.

In one embodiment, first safe operating distance 512 is 1000 microns, yielding a 12 micron resolution for 80 samples of electronic focus signal 115. In this embodiment, second safe operating distance 537 is 240 microns, that yields a 4 micron resolution for 80 samples of electronic focus signal 115.

Appendix E illustrates one embodiment of software code that implements in host workstation 116 various computations for the auto-focus operations described above for use with a low power objective lens. Routine lonui_StageAF in host workstation 116 initializes variables and then invokes routine lon_APIQuery that determines the characteristics of objective lens 110. Then host workstation 116 determines a travel limit fZLimUm and then finds out magnification iObjPwr of objective lens 110. Then host workstation 116 instructs fine Z-stage 120 to move to a predetermined middle position HDWR_FASTZ_MID_POS.

Host workstation 116 then saves values of several parameters currently in use, such as the intensity, system gain, zero value and Y axis scan amplitude. Then host workstation 116 sets a new zero value and a new system gain and invokes routine lonuiLoPwrStgAF for a low power objective lens or alternatively, performs various steps described below for a high power objective lens.

Routine lonuiLoPwrStgAF initializes variables, such as variable dHalfRange2ndPass and variable fZPos and then calls routine lonuiMoveStageZUm that moves target 112 to 1,000 microns below the uppermost travel limit set by first safe operating position 520 in one embodiment to starting position 516 (FIG. 6). Then host workstation 116 calls routine mot_SetCurrAxisSpeed that sets the target speed for travelling for one second. Next, host workstation 116 calls routine lonuiMoveStgAndReadInten to move coarse Z-stage 122 to a stopping position 522 and during this movement reads the intensity (i.e. magnitude) of electronic focus signal 115. Routine lonuiMoveStgAndReadInten returns with an estimate of focus position 203 and the peak intensity. Based on the peak focus signal intensity, host workstation 116 adjusts auto-focus gain for optimal focus signal intensity in the next pass. If the peak intensity is lower than optimal, auto-focus gain is increased, if peak intensity is higher than optimal, auto-focus gain is decreased.

Routine zr_SetTargetLaserIFrImgInten sets the auto-focus gain based on the intensity. Then host workstation 116 reads the current position by calling routine stg_ReadStageZUm.

After optimal auto-focus gain is set, host workstation 116 then determines target 112's position in second coarse pass 514 as variable fTarget. Variable fTarget is normally the Z position of target 1128 for the first estimate of focus position 203. If, during first coarse pass 513 electronic focus signal 115's intensity is all zero, or if photodetector 114 is overloaded, then variable fTarget is set to 400 micron away from coarse Z stage 122's position. Since actual focus position 203 can be at some distance away from the first estimate of focus position 203 the second coarse pass 514 starts at some distance before the first estimate and ends at some distance after the first estimate.

The distance before and after the first estimate is chosen to be several times the depth-of-focus of objective lens 110 (in this embodiment 120 microns) and is contained in variable dHalfRange2ndPass. If after first coarse pass 513, coarse Z stage 122 is at a position greater than distance dHalfRange2ndPass away from estimate of focus position 203, then host workstation 116 instructs coarse Z stage 122 to move to a starting position at a distance dHalfRange2ndPass before first estimate of focus position, and second coarse pass 514 starts from this position. Otherwise, second coarse 514 starts from wherever first coarse pass 513 ends. The stopping position of second coarse pass 514 is calculated to be two times dHalfRange2ndPass away from starting position 514.

Then host workstation 116 changes the speed so as to move through second safe operating distance 537 within one second, by using routine mot_SetCurrAxisSpeed. Host workstation 116 then moves target 112 through second safe operating distance 537 by calling routine lonuiMoveStgAndReadInten, and computes the second estimate of focus position 203.

After computing host workstation 116 moves target 112 to the second estimate by calling routine lonuiMoveStageZUm.

After one of the previously described coarse auto-focus operations is completed, an optional fine auto-focus operation can be performed, if electronic focus signal 115 has a sharp peak, as illustrated in FIG. 3A. The zero position offset 310 and photodetector gain illustrated in FIG. 3B are not utilized during a fine auto-focus operation. During a fine auto-focus operation, target 112 is moved by a fine Z-stage 120 that includes a piezoelectric element 1130. Fine Z-stage 120 is described in detail later.

Before a fine auto-focus operation is performed, target 112 is positioned by a coarse auto-focus operation (above) such that focus position 203 is within the operating range of fine Z-stage 120 (FIG. 1). In one embodiment, fine Z-stage 120 has an operating range of 50 microns along the Z-axis. Consequently, by performing the coarse auto-focus operation described above, in which target 112 is positioned within +/- 10 microns of the focus position 203, target 112 is positioned such that focus position 203 is within the operating range of fine Z-stage 120.

FIG. 7 is a graphic representation of one embodiment (henceforth "first" embodiment) of a fine auto-focus operation as encoded in routine AFocusServo on page 68 of microfiche appendix A. Although FIG. 7 illustrates two fine passes, any number of passes other than two can be used in other embodiments. The vertical axis in FIG. 7 illustrates the position of target 112 along the Z-axis. The horizontal axis in FIG. 7 illustrates the magnitude of electronic focus signal 115.

Prior to a first fine pass 601, microprocessor 403 sends a zeroing signal to position control register 408 (FIG. 4). This signal is transmitted through DAC 409, summing node 410 and amplifier 411 to piezoelectric element 1130 (FIG. 11) in fine Z-stage 120. Fine Z-stage 120, which was positioned in the middle of its operating range during the coarse auto-focus operation, moves to the lowermost position 619 (FIG. 7) of its operating range (in a startup pass similar to that described above for a coarse auto-focus operation) in response to the zeroing signal. Microprocessor 403 then instructs control register 406 to transmit a reset signal to clear flip flop 402. This instruction allows flip flop 402 to detect when electronic focus signal

115 exceeds threshold value 304.

To begin first fine pass 601, microprocessor 403 transmits a series of signals to position control register 408, thereby causing fine Z-stage 120 to move target 112 in positive Z direction 631 at a relatively high velocity. In one embodiment, this velocity is approximately 75 microns per second.

While it is necessary to precisely control the movement of target 112 during first fine pass 601, piezoelectric element 1130 (FIG. 11) of fine Z-stage 120 has a slightly non-linear position response to the voltage supplied by amplifier 411 (FIG. 4). To correct for this non-linear characteristic, a proximity sensor 1135 (FIG. 11) in fine Z-stage 120 produces an electrical feedback signal that is transmitted to summing node 410 (FIG. 4) and subtracted from the output signal of DAC 409 to create an error signal. When integrator 420 receives any non-zero error signal, integrator 420 generates an output signal that forces the error signal to zero. In this manner, integrator 420 compensates for the non-linear response of the piezoelectric element 1130, thereby allowing for linear control of the target's position by piezoelectric element 1130.

First fine pass 601 uses the threshold method so that before target 112 reaches the top of the operating range 620 of fine Z-stage 120, electronic focus signal 115 exceeds the threshold value 304 at a first trip position 604, because, as described above, the coarse auto-focus operation ensures that focus position 203 lies within the range of fine Z-stage 120 (FIG. 7). At this time, flip flop 402 latches, thereby enabling a bit in status register 405. Microprocessor 403, which continuously monitors status register 405, detects that the change in status of flip flop 402 and signals position control register 408 to stop movement of fine

Z-stage 120, and thereby stop movement of target 112. Target 112 comes to rest at a first stopping position 605 which can either be above or below focus position 203.

Consequently, prior to performing a second fine pass 615, microprocessor 403 instructs fine Z-stage 120 to reposition target 112 a predetermined fixed distance 614 in negative Z direction 632 such that a second stopping position 608 of target 112 is below focus position 203. Factors that must be considered when selecting distance 614 are similar or identical to the factors described above regarding the distance required to stop target 112 during first coarse pass 513. In one embodiment, distance 614 is 3.6 microns.

After target 112 has been repositioned at second stopping position 608, a second fine pass 615 is performed by moving target 112 in positive Z direction 631 at a relatively low velocity (e.g. 7.5 microns per second). During second fine pass 615, microprocessor 403 monitors the output of ADC 407, rather than the status of flip flop 402. The output of ADC 407 is a digital representation of electronic focus signal 115. In this manner, microprocessor 403 measures the value of electronic focus signal 115 as target 112 moves along the Z-axis.

Second fine pass 615 uses a peak detection method in which software within microprocessor 403 maintains an updated record of the maximum value output by ADC 407 and the position of target 112 at this maximum output value. Upon detecting an increasing value of electronic focus signal 115, followed by a decreasing value of electronic focus signal 115, microprocessor 403 instructs fine Z-stage 120 to stop the motion of target 112 at third stopping position 611. Third stopping position 611 is located above focus position 203 because of target overshoot which occurs for reasons similar to those described above. Microprocessor 403 then instructs fine Z-stage 120 to move target 112 in negative Z direction 632 to position 612 where the maximum value output by ADC 407 was detected.

In one embodiment, position 612 is typically within one tenth of a micron of focus position 203. This accuracy is determined by the velocity of target 112 and the number of bits used in ADC 407. The slower the velocity of target 112 during second fine pass 615 and the greater the number of bits used in ADC 407, the closer position 612 will be to focus position 203. The entire fine auto-focus operation is completed in approximately one second in one embodiment. The completion time is dependent upon the location of focus position 203 within the range of fine Z-stage 120. Computer code used to perform an auto-focus operation in accordance with one embodiment of fine auto-focus method is set forth in Appendix A. The computer code of Appendix A is written in neuron C language, which requires an ECHELON compiler, available from Echelon Corp. of 4015 Miranda Ave., Palo Alto, CA 94304.

FIG. 8A is a graphic representation of another embodiment (henceforth "second" embodiment) of a fine auto-focus operation. Although FIG. 8A illustrates two fine passes, a number of passes other than two can be used in other embodiments. The vertical axis in FIG. 8A illustrates the position (e.g. elevation) of target 112 along the Z-axis. The horizontal axis in FIG. 8A illustrates the magnitude (i.e. strength) of electronic focus signal 115. In first fine pass 750, target 112 is moved to each of positions 701-732. In second fine pass 850, target 112 is moved to each of positions 801-832.

Prior to first fine pass 750, microprocessor 403 sends a zeroing signal to position control register 408

(FIG. 4). This signal is transmitted through DAC 409, integrator 420, summing node 410 and amplifier 411 to piezoelectric element 1130 of fine Z-stage 120. Fine Z-stage 120, which was positioned in the middle of its operating range before a coarse auto-focus operation (e.g., Step No. 2048 in FIG. 8A), moves to the bottom of its operating range (i.e., Step No. 0 in FIG. 8A) in response to the zeroing signal, as illustrated by FIG. 8B.

First fine pass 750 uses a peak detection method. During first fine pass 750, target 112 is moved in the positive direction, e.g. upward through the full range of motion of fine Z-stage 120 (i.e., 50 microns). As shown in FIG. 8A, this range is divided into 4096 steps. Other numbers of steps can be used in other embodiments. Microprocessor 403 can position target 112 at any one of these steps by sending a digital word to DAC 409 (FIG. 4). During first fine pass 750, microprocessor 403 sequentially provides 32 digital words to DAC 409, causing fine Z-stage 120 to sequentially move target 112 to each of 32 positions 701-732. Each of the 32 positions 701-732 are separated by 128 steps (approximately 1.56 microns in one embodiment). At each of the 32 positions 701-732, the output voltage of ADC 407 (corresponding to the electronic focus signal 115) is repeatedly measured and digitally filtered (low-pass) by microprocessor 403 to obtain a single value for each of the 32 positions 701-732. Microprocessor 403 saves the peak value of the output voltage and the position at which the peak value occurred.

Because the 32 positions 701-732 of first fine pass 750 are spaced 1.56 microns apart, and electronic focus signal 115 has a depth of focus 302 of approximately 2.54 microns, a focus position is not missed. A relative peak value of the electronic focus signal is found within the 32 positions 701-732 as long as there is enough system gain to distinguish electronic focus signal 115 from background value 303 and enough system gain such that electronic focus signal 115 can be detected by ADC 407. System gain is determined by the laser power, the gain of photodetector 114, and the reflectivity of the sample. The present invention therefore has an advantage over prior art auto-focus microscopes, which require a much larger system gain in order for an auto-focus operation to be performed.

The position 718 at which the peak value is detected during first fine pass 750 (illustrated as Step No. 2304 in FIG. 8A) becomes the center of the second fine pass 850. Microprocessor 403 instructs fine Z-stage 120 to move target 112 to 128 steps below position 718 (i.e., to Step No. 2176). In other embodiments, a different number of steps can be used.

Next, microprocessor 403 sequentially provides 32 digital words to DAC 409, causing fine Z-stage 120 to sequentially move target 112 upward to each of 32 positions 801-832. Each of 32 positions 801-832 are separated by 8 steps (approximately 0.0977 microns). The total distance of second fine pass is 3.125 microns. At each of 32 positions 801-832, output voltage of ADC 407 (corresponding to electronic focus signal 115) is repeatedly measured and digitally filtered (low-pass) by microprocessor 403 to obtain a single value for each of 32 positions 801-832. Microprocessor 403 saves the peak value of the output voltage and the position at which the peak value occurred.

In yet another embodiment (henceforth "third" embodiment), first fine pass 750 is performed as described above for the second embodiment and second fine pass 850 is performed by positioning target 112 at

128 steps above position 718 sample value was detected during first fine pass 750 (e.g., at Step No. 2432 in FIG. 8A and at position 852 in FIG. 8C). Microprocessor 403 then sequentially provides 32 digital words to DAC 409, causing the fine Z-stage 120 to sequentially move target 112 in the same direction (e.g. downward) through 32 positions 832-801 in FIG. 8A. (See also FIG. 8C.) Moving in the same direction in second fine pass 850 avoids the large acceleration caused by direction reversal at position 851 (FIG. 8B).

After passing through the entire range (3.125 micron in one embodiment) of second fine pass 850, microprocessor 403 instructs fine Z-stage 120 to position target 112 at position 812 at which a peak value is detected during second fine pass 850 (illustrated as Step No. 2272 in FIG. 8A). At the end of second fine pass 850, target 112 is positioned adjacent to actual focus position 203, i.e., within half the length between each of the 32 positions of the second fine pass 850 (within at least 0.0488 microns in one embodiment).

An advantage of a second fine pass (and additional fine passes) that target 112 is positioned quickly and reliably without relying on a threshold value. In one embodiment, fine Z-stage 120 positions target 112 sixty-seven times (including sixty-four positions at which measurements are taken and three positions at which measurements are not taken) to perform the fine auto-focus operation in approximately 0.9 seconds. Computer code used to perform one embodiment an auto-focus operation for a fine pass is set forth in microfiche appendix A.

A preferred embodiment of fine Z-stage 120 is shown in FIGs. 9-11. Appendix D at pages 127 and 128 lists various parts used in the embodiment shown in FIGs. 9-11. Referring to FIG. 9, fine Z-stage 120 includes a square bottom plate 1010, four stationary bars 1020 fixedly connected along the edges of the bottom plate 1010, four rotating bars 1030 pivotally connected to the stationary bars 1020 such that each rotating bar 1030 is connected to one stationary bar 1020, and a square top plate 1050 pivotally connected along its edges to the four rotating bars 1030.

A first set of horizontally-disposed flexures 1040 is connected between upper surfaces of the stationary bars 1020 and the rotating bars 1030, and a second set of vertically-disposed flexures 1060 is connected between side surfaces of the rotating bars 1030 and the edges of the top plate 1050. In addition, the fine Z-stage 120 includes a piezoelectric actuator mechanism 1100 disposed in a space formed between the top plate 1050 and the bottom plate 1010. Finally, an optional biasing spring 1140 is connected between the top plate 1150 and the bottom plate 1110 for biasing the top plate 1150 toward the bottom plate 1110.

Referring to FIG. 10, the bottom plate 1010 is preferably a flat aluminum sheet 8 by 8 inches wide and 0.375 inch thick. As shown in FIG. 10, the bottom plate 1010 includes a receiving hole 1011 within which is located a pin 1012 for securing a first end of the optional biasing spring 1140. The bottom plate 1010 also includes an upper surface 1013.

Referring back to FIG. 9, the stationary bars 1020 are preferably aluminum bars 5.5 inches long, 0.65 inch high and 0.5 inch wide. The stationary bars 1020 are connected to the upper surface 1013 of the bottom plate 1010 using screws. The stationary bars 1020 are formed into a square frame and located along the outer edges of the bottom plate 1020. Referring again to FIG. 10, each stationary bar 1020 includes an upper surface 1021. A lip 1022 is formed along an outer edge of the upper surface 1021 of each stationary bar 1020.

The rotating bars 1030 are preferably aluminum bars which are 5.5 inches long, 0.5 inch high and 0.6 inch wide. The rotating bars 1030 are pivotally connected to the stationary bars 1020, each rotating bar 1030 being connected to one stationary bar 1020. Each rotating bar 1030 includes an upper surface 1031 and an inner side surface 1032. A lip 1033 is formed along a lower edge of the inner side surface 1032 of each rotating bar 1030.

As shown in FIG. 10, thin horizontally-disposed flexures 1040 are connected between the stationary bars 1020 and the rotating bars 1030. Each flexure 1040 is a thin sheet of 303 stainless steel which is 1 inch long, 1 inch wide and approximately 0.01 inch thick. Each flexure 1040 has one portion connected to the upper surface 1021 of a stationary bar 1020 by a first fixture 1041, a second portion connected to the upper surface 1031 of a rotating bar 1030 by a second fixture 1042, and a small pivot portion 1043 located between the stationary bars 1020 and the rotating bars 1030. Two flexures 1040 are connected between each stationary bar 1020 and its associated rotating bar 1030, thereby restricting each rotating bar 1030 to pivot around the pivot portion 1043 such that the rotating bar 1030 remains in a parallel relationship with its associated stationary bar 1020.

The top plate 1050 is preferably a flat aluminum sheet 5.5 by 5.5 inches wide and 0.5 inch thick. Referring to FIG. 10, the top plate 1050 includes a receiving hole 1051 within which is located a pin 1052 for securing a second end of the optional biasing spring 1140 (discussed below). In addition, the top plate 1050 defines threaded holes 1053 (only one shown) for receiving preload screws 1054 and 1055 (see FIG. 9). Finally, the top plate 1050 includes side surfaces 1056.

As shown in FIG. 10, thin vertically-disposed flexures 1060 are connected between the top plate 1050 and the rotating bars 1030. Similar to the above-described horizontally-disposed flexures 1040, each vertically-disposed flexure 1060 is a thin sheet of 303 stainless steel which is 1 inch long, 1 inch wide and 0.01 inch thick. Each vertically-disposed flexure 1060 has one portion connected to the inner side surface 1032 of a rotating bar 1030 by a fixture 1061, a second portion connected to a side surface 1056 of top plate 1050 by a fixture 1062, and a pivot portion 1063 located between the rotating bars 1030 and the top plate 1050. Two flexures 1060 are connected between each rotating bar 1030 and one edge of the top plate 1050, thereby restricting the top plate 1050 to pivot with respect to the rotating bars 1030 such that the top plate 1050 remains in a parallel relationship with the rotating bars 1030.

Referring to FIGs. 10 and 11, the piezoelectric actuator mechanism 1100 includes a retainer block 1110 connected to the bottom plate 1010, a rotating block 1120 integrally and pivotally connected to the retainer block 1110, and an extending portion 1114 integrally and pivotally connected to the retainer block 1110. Further, a piezoelectric element 1130 is received in the retainer block 1110 and has a free end contacting the rotating block 1120. Finally, a sensor 1135 is received in the retainer block 1110 and generates a signal corresponding to an amount of rotation of the extending portion 1114.

Referring to FIG. 11, the retainer block 1110 is formed from 7075-T6 high-strength aluminum alloy and is approximately 1.1 inch long, 0.6 inch wide and 0.1 inch thick. The retainer block 1110 defines a first through-hole 1111 for receiving the piezoelectric element 1130. A stainless steel plate 1112 is connected by fasteners 1113 to retain the piezoelectric element 1130 within the through-hole 1111. In addition, the retainer block 1110 defines a second through-hole 1117 for receiving the sensor 1135. The retainer block 110 also includes a hole 1118 for receiving a sensor lock screw 1119 which contacts and secures the sensor 1135.

As shown in FIG. 11, an extended portion 1114 is formed from 7075-T6 high-strength aluminum alloy and is integrally connected to the upper surface of the retainer block 1110 by a thin flexure. The extended portion 1114 includes a socket 1115 for receiving a 0.125 inch diameter steel ball 1116 which contacts the top plate 1150. The extended portion 1114 is preloaded downward by the preload screw pressing against the steel ball 1116 (see FIG. 9). The sensor 1135 transmits a signal representing a distance between the sensor and a side wall of the extended portion 1114 which varies in response to the rotation of the extended portion 1114. The signal is used to determine the vertical displacement of the top plate 1050, as discussed above.

As shown in FIGs. 10 and 11, the rotating block 1120 is a prism formed from 7075-T6 high-strength aluminum alloy and is approximately 1 inch long. As shown in FIG. 10, the rotating block 1120 includes a vertical side wall 1121, an upper wall 1122 and a diagonal wall 1123. The rotating block 1120 is integrally connected to the upper surface of the retainer block 1110 by a flexure 1124. The side wall 1121 defines a socket 1125 for receiving a 0.125 inch diameter steel ball 1126 which contacts an end of the piezoelectric element 1130. In addition, the upper wall 1122 defines a second socket 1127 for receiving another 0.125 inch diameter steel ball 1128 which contacts the preload screw 1154 mounted in the top plate 1050. It is noted that the steel ball 1128 is located further from the flexure 1124 than the steel ball 1116, which is mounted on the extended portion 1114.

The piezoelectric element 1130 is a cylindrical unit housed in the through-hole 1111 of the retainer block 1110 such that movement of a first end is prevented by the plate 1112, and a second end contacts the vertical side wall 1121 of the rotating block 1120 through the ball 1126. The piezoelectric element 1130 is connected to amplifier 411, as discussed above. A preferred piezoelectric element is sold by Physic Instrument of Waldbronn, Germany under model number P830.20.

The sensor 1135 is also a cylindrical unit housed in the through-hole 1117 and spaced a predetermined distance from the vertical side wall of the extended portion 1114. The sensor 1135 is connected to summing node 410, as discussed above. Once the sensor 1135 is mounted a predetermined distance from the side wall of the extended portion 1114, the sensor lock screw 1118 is tightened against the side of the sensor to prevent movement of the sensor 1135 within the through-hole 1117. A preferred sensor is sold by Kaman Instrumentation of Colorado Springs, CO under model number SMU 9000-15N.

Finally, an optional biasing spring 1140 may be connected between the pin 1012 formed in the bottom plate 1010 and the pin 1052 connected to the top plate 1050.

In operation, when no actuating voltage is applied to the piezoelectric element 1130, the optional biasing spring 1140 pulls the top plate toward the bottom plate until the top plate abuts and rests against the ball 1116. In this position, the upper surface 1021 of the stationary bars 1020 and the upper surface 1031 of the rotating bars 1030 are aligned such that the horizontal flexures 1040 are substantially planar. In addition, the inner side surfaces 1032 of the rotating bars 1030 and the side surfaces 1055 of the top plate 1050 are aligned such that the vertical flexures 1060 are substantially planar.

Upon application of an actuating voltage across the piezoelectric element 1130, the piezoelectric element 1130 presses against the side wall 1121 of the rotating block 1120 through the ball 1126, thereby causing the rotating block 1120 to rotate about a pivot portion 1062 of the flexure 1060 connecting the rotating block 1120 to the retainer block 1110. As the rotating block 1120 is rotated away from the retainer block 1110, the ball 1128 presses upward on the preload screw 1154, causing the top plate 1050 to move upward.

Upward movement of the top plate 1050 causes a rotation of the rotating bars 1030. As the rotating bars 1030 are rotated, the horizontally disposed flexures 1040 restrain the rotating bars 1030 such that they remain parallel with their associated stationary bars 1020. In addition, the vertically disposed flexures 1060 cause the top plate 1050 to remain parallel with each of the rotating bars 1030. As a result, the rotating bars 1030 act as torsion bars which prevent unwanted rotation or translation of the top plate 1050, there by resulting in the upper surface of the top plate 1050 remaining parallel with the bottom plate 1010.

Returning to FIG. 1, the mirror control 124 of FIG. 1 is used to rotate X-mirror 106 and Y-mirror 108 such that laser beam 123 can scan more than a single point on target 112 while performing the coarse and/or fine auto-focus operations. Thus, if X-mirror 106 is rotated while Y-mirror 108 is held still, laser beam 123 will trace a line along the X-axis on target 112.

Similarly, X-mirror 106 can be held still while Y-mirror 108 is rotated, thereby tracing a line along the Y-axis on target 112.

In one embodiment of the present invention, the coarse and fine auto-focus operations are performed by scanning a line, rather than a spot, on target 112. In this embodiment, the line scan is performed at a frequency of approximately 8 Khz while target 112 is held stationary at one of several elevations in the range of movement along the Z-axis.

For an area scan method, X-mirror 106 and Y-mirror 108 can both be rotated to trace either a small area or selected parts of a larger area in the X-Y plane of target 112. The small area can be an area in the center of the field of view, while the larger area can be the entire field of view. During the area scan, microscope system 100 covers various features in the field of view in a path similar to a raster path of a television tube. To generate an electronic focus signal at a current Z-axis elevation, an area peak detector 1210 records and a microprocessor 403 reads the value generated by the highest amount of reflected light 123R that occurs during the area scan.

By using a line scan method or an area scan method and averaging the result of the scan to create electronic focus signal 115, the estimate of focus position 203 during a coarse auto-focus operation or a fine auto-focus operation becomes less sensitive to local height variations on the surface of target 112.

In a preferred embodiment, the area scan method is used for a fine auto-focus operation, although an area scan can also be used for a coarse auto-focus operation. In this embodiment, area peak detector 1210 (FIG. 12) is synchronously reset at the start of the area scan (anywhere on the area can be used as the start, as long as the reset is synchronous). FIG. 12

is similar or identical to FIG. 4, except for certain components that are described below. Area peak detector 1210 acquires or "loads up" during the rest of the area scan as follows. Area peak detector 1210 has a storage capacitor C67 (FIG. 14) that is discharged when area peak detector 1210 is reset. The voltage on storage capacitor C67 is continuously compared to the input voltage of op-amp U26. When op-amp U26's input voltage exceeds the stored voltage of capacitor C67, capacitor C67 is charged up to equal the input voltage. This process happens continuously, so that storage capacitor C67 records the highest input voltage to op-amp U26 no matter how short its duration, (within bandwidth limitations) that occurs after the reset. The only way that storage capacitor C67 discharges is through the reset mechanism or through leakage currents.

Just before being reset for the next area scan, a sampling ADC 5021 digitizes the output of area peak detector 1210. The output of ADC 5021 represents electronic focus signal 115 for the current elevation of target 112. ADC 5021 has a track-and-hold buffer in front (in one embodiment buffer is part of the architecture of ADC 5021), so that acquiring the output of area peak detector 1210 is virtually instantaneous. The ADC can digitize a voltage at the output of area peak detector 1210 at 0.386 V/μs without error, that translates to an acquisition time of about 25 ns. Area peak detector 1210 can be reset very quickly, using for example, a 500 nanosecond pulse. The time during which output of area peak detector 1210 is digitized and area peak detector 1210 reset is an extremely small portion of the overall area scan time so that only a very small portion of the area (e.g. 0.15%) is ignored while area peak detector 1210 is reset.

Normally an X-axis laser scanner (not shown) (also referred to as "line scanner") that is used in microscope system 100 is resonant, while a Y-axis scanner (not shown), is a closed-loop galvo scanner that follows a "sawtooth" waveform (relatively slow scan followed by a fast flyback) during normal imaging that is unrelated to an auto-focus operation.

For an area scan used to find focus position 203, the Y-axis scanner (also referred to as "page scanner") follows a triangular wave profile 1301 (FIG. 13A) that is different from the "sawtooth" waveform at a much higher scan rate (e.g. 125 Hz) than normal (e.g. 13 Hz). The high scan rate allows microscope system 100 to trace a raster (e.g. sine wave) path in the area in the field of view quickly (8msec in one embodiment), albeit at less resolution (e.g. 32 lines/frame). At the end of an auto-focus operation, once target 112 has been positioned at focus position 203, the page scanner is returned to its normal slow sawtooth waveform.

Buffer 1230 connected to the output terminals of ADC 5021 allows signals on ADC 5021's output terminals to be turned on and held on without interfering with the operation of data bus 5006. This is required because ADC 5021 used in this embodiment cannot be configured to do a conversion without turning on signals at its output terminals. Conversion is initiated by hardware at a very precise predetermined time, at the end of an area scan. Signals that result from conversion remain active at the output terminals of ADC 5021 until microprocessor 403 detects that a scan is complete, reads ADC 5021 through buffer 1230 by asserting signal R_ADC* and resets ADC 5021 by pulsing signal RST-ZSYNC*.

FIG. 13B illustrates events after the rising edge of signal ZSYNC. Signal ZSYNC rises at the end of each page scan to indicate completion of a scan cycle. The time scale in FIG. 13B is different from that of FIG.

13A. Signal ZSYNCD*, that is a saved version of signal ZSYNC, goes low in response to a rising edge in signal ZSYNC. At the same time, signal CNVRT* (FIG. 13B) also goes low, initiating an analog to digital conversion and causing signal ADC_BSY* (FIG. 13B) to go low. Signal BUSY1* and signal BUSY2* are delayed versions of signal ADC_BSY*, and are used to generate an active high pulse in signal RST_PKDET (FIG. 13B) after signal ADC_BSY* returns high, to reset peak detector 1220 after ADC's conversion is completed. So ADC 5021 automatically starts a conversion when signal ZSYNC goes high, and area peak detector 1210 is automatically reset when the conversion is complete. Because there is a T/H in ADC 5021, the peak detector 1220 can be reset after the T/H has acquired, and before the conversion is complete.

Signal CNVRT* stays low until signal RST_ZSYNC* falls, which keeps the conversion result present at the output terminals of ADC 5021. Software in microprocessor 5000 reads ADC 5021 (via signal R_ADC*) before resetting signal ZSYNCD*. Signal ZSYNCD* is an input signal to microprocessor 5000 and indicates that a frame is complete.

In one embodiment, there is a relatively long and unpredictable latency in the response of host workstation 116 to signal ZSYNCD*. Hence most of the control signals (e.g. signals RST_PKDET and CNVRT*) for area peak detector 1210 and ADC 5021 are generated in hardware.

Area peak detector 1210 consists primarily of op-amps U26 and U27 (FIG. 14). The "hold" capacitor for area peak detector 1210 is C67. When the input voltage at pin 3 of op-amp U26 is greater than the hold voltage at capacitor C67, D9 is reverse-biased, so op-amp U26 has no feedback (at least momentarily). The output of op-amp U26 will therefore rise until op-amp U27 responds to the increased output of op-amp U26 and provides feedback via resistor R47. In this manner, when the input voltage to op-amp U26 is greater than the hold voltage, op-amp U26 drives the hold voltage higher to equal the input voltage. When the input voltage to op-amp U26 is less than the hold voltage, a diode D12 is reverse-biased and op-amp U26 receives a feedback signal through diode D9. Because diode D12 is reverse-biased, changes in the input voltage at op-amp U26 do not affect the hold voltage when op-amp U26's input voltage is less than the hold voltage.

Op-amp U27 buffers the hold voltage to minimize the current drawn from hold capacitor C67 (and hence the droop rate), and to provide feedback to op-amp U26. Transistor Q2 resets hold capacitor 67. Transistor Q1 guarantees that the input voltage of op-amp U26 is less than the hold voltage during reset (the hold voltage is zero during reset), so that transistor U26 does not slew positive during the reset operation. When the input voltage at U26 pin 3 is greater than the hold voltage at C67, D9 is reverse biased, so U26 has no feedback (at least momentarily). U26's output rises (slews positive) at the maximum rate possible for the particular op-amp (its "slow side") until U27 responds to the increased output of U26 and provides feedback via R47.

Diode D10 and resistor R46 establish a small negative voltage (approx. -0.4V in one embodiment) for clamping the input signal during reset. Resistors R53 and R47 slow down area peak detector 1210, to reduce overshoot that is common in conventional area peak detectors. Resistor R53 limits the rate at which capacitor C67 can be charged, resulting in an acquisition bandwidth of 2.3 MHz in one embodiment. Resistor R47 works with parasitic capacitance of U26 and D9 to form a feedback filter for composite op-amp

U26/U27. Resistor R47 acts to stabilize (reduce overshoot) of the composite op-amp. Both resistors R53 and R47 are determined empirically. Even with resistors R53 and R46, area peak detector 1210 provides an extremely fast response, and the combination of resistor R52 and capacitor C64 acts as a filter to limit the bandwidth of the input signal, so that high frequency noise does not pass through to area peak detector 1210.

An auto-focus routine AFPeakSync (page 98 of microfiche appendix B) that performs an autofocus operation during using an area scan method by a microscope system 100 is similar to auto-focus routine AFFast (page 97 of microfiche appendix B) described above (FIG. 8B). As seen in FIG. 8B, in a first fine pass, auto-focus routine AFFast steps from bottom to top of the fine Z-axis range of movement in large steps, then returns to a position below the coarse step that had the highest focus signal.

In a second fine pass, auto-focus routine AFFast then steps upward again, but this time in small steps (e.g. 0.098 μm). Auto-focus routine AFFast causes a large acceleration due to direction reversal just before the second fine pass by stepping to a position just below the estimated focus position (FIG. 8B). [Larger accelerations imply larger position errors in fine Z-stage 120 and are preferably avoided to improve positioning fidelity.]

Auto-focus routine AFPeakSync also uses two fine passes: a first fine pass and a second fine pass. The first fine pass of routine AFPeakSync (henceforth "AFPeakSync first fine pass") differs from the coarse pass of the coarse Z-stage in the manner described below. The AFPeakSync first fine pass operates similar to the coarse pass of auto-focus routine AFFast. There is no synchronization, and auto-focus routine

AFPeakSync steps and measures as quickly as possible, using large steps (e.g. 1.56 micrometer).

Software (Appendix F) in host workstation 116 sets up the page scanner (not shown) to perform the area scan method during the AFPeakSync first fine pass. Routine lonuiSuperfineAF sets up all parameters for use in the area scan method. In routine lonuiSuperFineAF, function lonui_Get_IndexFrPixZoomEnum returns with the current X and Y scanner parameters, such as amplitude of oscillation. After the scanner parameters are set, host workstation 116 instructs fine Z-axis controller 118 to execute a fine auto-focus operation. After the fine auto-focus operation is done, host workstation 116 returns scanner parameters to their original values.

In alternative embodiments, the page scanner can execute a line scan method or a spot method during the AFPeakSync first fine pass. The AFPeakSync first fine pass is merely used to move target 112 close enough to focus position 112 so that a AFPeakSync second fine pass can correct residual errors.

Auto-focus routine AFPeakSync avoids the large acceleration of auto-focus routine AFFast in a second fine pass by stepping to a position above the estimated focus position at the end of a first fine pass and continuing in small steps for the second fine pass in the same direction, as illustrated by FIG 8C.

In one embodiment, a second fine pass in auto-focus routine AFPeakSync covers twice the range of a fine pass in auto-focus routine AFFast, which reduces sensitivity to manufacturing variations and improves performance of the autofocus operation. The increased range of the AFPeakSync second fine pass compensates for any position errors in the AFPeakSync first fine pass that could result in a AFPeakSync step with the highest focus strength being off by one rough step.

Auto-focus routine AFPeakSync spends two page scanner cycles at each position of the AFPeakSync second fine pass. The first cycle allows the fine Z-axis (signal ACTUAL Z POSITION in FIG. 13C) to settle out, and the second cycle acquires the peak magnitude of electronic focus signal 115 from the page scan at the current elevation. Another embodiment of an auto-focus routine can spend only one page scanner cycle by allowing the fine Z movement to settle during a trace 1310 and by acquiring the peak during the retrace 1320.

Auto-focus routine AFPeakSync executes in about 1.5 seconds, compared to 0.9 seconds for auto-focus routine AFFast because of the time required to perform an area scan at each of several elevations of target 112. In spite of such an increased execution time, a microscope system that uses auto-focus routine AFPeakSync is faster overall as compared to a microscope system that uses auto-focus routine AFFast, because with increased reliability of the auto-focus operation using routine AFPeakSync, manual focus adjustments are needed only rarely.

The first fine pass of auto-focus routine AFPeakSync steps as quickly as possible irrespective of where the scanner is during sampling. In the second fine pass, however, auto-focus routine AfPeakSync runs the page scanner at a high rate (>100Hz) and allows one page scanner cycle for fine Z-stage 120 to settle, before doing data acquisition on the next cycle, as illustrated in FIG. 13C. Auto-focus routine AFPeakSync therefore spends two page scanner cycles at each Z position (e.g. elevation). Furthermore, in area peak detector mode, an analog-to-digital converter 5021 is automatically started when a rising edge in signal ZSYNC occurs, and area peak detector 1210 is automatically reset. So, auto-focus routine AFPeakSync causes host microprocessor 5000 to wait for a rising edge in signal ZSYNC and then read the peak magnitude detected from the previous page scan from analog-to-digital converter 5021. Resetting signal ZSYNC also resets analog-to-digital converter 5021 and so ADC 5021 is read before resetting signal ZSYNC.

In peak detector mode, analog-to-digital converter 5021 is controlled only by a rising edge in signal ZSYNC. An asynchronous conversion is not possible without resetting the peak detector mode bit. Peak detector 1210 obtains a representation of the highest intensity that occurred in a video frame at a given Z elevation. Peak detector 1210 allows microscope system 100 to repeatably focus on the layer with the highest reflectivity.

Microscope system 100 can use programmable offset FocOff to allow the user to select any predetermined layer to focus on. In one embodiment, microscope system 100 initially focuses on a layer ("brightest layer") of target 112 that generates the largest electronic focus signal. The user can adjust the position of target 112 to bring a predetermined layer into focus. Microscope system 100 records the target's adjustment as an offset from the brightest layer. In subsequent auto-focus operations, microscope system 100 focuses on the brightest layer and then automatically moves target 112 through the adjustment offset that was specified by the user, to focus on the predetermined layer.

As the relative reflectivities of the layers in a wafer are repeatable and independent of the illumination power, the relative positions of a predetermined layer chosen by a user is at a repeatable offset from the brightest layer. So microscope system 100 can repeatably focus on any predetermined layer after a single adjustment. Microscope system 100 focuses on such a predetermined layer as long as the layer structure stays constant, for example for array type structures, such as RAM arrays. If, after selecting a predetermined layer, areas off a RAM array are to be imaged, a simple re-adjustment allows the user to continue.

In one embodiment, the function to specify the offset is implemented as a system function that can be mapped by a user to any function key of host workstation 116. Once mapped to a function key, the offset function can be accessed by the user from the keyboard at any time. When so accessed, host workstation 116 compares the current target position with the focus position estimate of last auto-focus operation, calculates the difference and stores the difference as the offset for finding the predetermined layer in future auto-focus operations.

The offset is limited to +/- 5 microns, to make it easier to recover from an improperly set offset (a typical semiconductor topology is less than 1 μm) and only fine Z-stage 120 is moved during offset calculation. The offset is calculated assuming coarse Z-stage 122 has not moved since last auto-focus operation.

While the present invention has been described in connection with specific embodiments, variations on these embodiments will be obvious to those having ordinary skill in the art. For example, target 112 may be moved by means other than a stepper motor or a piezoelectric element, such as a linear voice coil motor or an electrostrictive actuator. Furthermore, target 112 can initially be positioned above the focus position, with the first pass in any of the embodiments beginning by moving target 112 in a negative Z direction. In addition, although the present invention was described in connection with a microscope that reflects a maximum intensity to the photodetector during a focused condition, it is clear that the invention may be modified to operate with a microscope that reflects a minimum intensity to the photodetector during a focused condition.

Furthermore, although the invention, as described, utilizes a laser beam 123 to perform both the auto-focusing and imaging operations, it is understood that a confocal laser optical system could be used to perform an auto-focusing operation for a non-confocal microscope system which utilizes only a white light source to perform the imaging operation. Such an application is advantageous because a non-confocal microscope system utilizing a white light imaging source results in a focus signal which is sinc function, rather than a sinc squared function. Because the sinc function does not exhibit a peak which is as pronounced as the sinc squared function, it is more difficult to determine the focus position using a focus signal generated by a non-confocal, white light optical system. Therefore, using a confocal laser optical system to generate the focus signal used to perform the auto-focus operation in a white light microscope results in a superior electronic focus signal, thereby allowing the auto-focus operation to be performed with greater precision. Confocal white light can also be used, with scanning replaced by detector array, such as a camera.

Although a certain number and types of passes are described above, other number and types of passes can also be used. For example a microscope system 100 can perform a single pass in a coarse auto-focus operation using a median point method and adjust the gain of photodetector 114 followed by two passes in a fine auto-focus operation to estimate focus position 203.

Various embodiments of the invention described above are encompassed by the attached claims.

APPENDIX A

```
// Fast Z-Axis Controller (P/N 000491) Test Program

// Source File:     Fastz.nc
// Author:          Chris F.
// Target Board:    000491, Fast Z-Axis Controller
// Target Socket:   M1-U2
// Ultrapointe Corp.
// PROPRIETARY! ALL RIGHTS RESERVED
// **** FOR INTERNAL USE ONLY ****
//

// The FastZ PCB controls a piezo-driven stage with analog position
// feedback.  The position command
// is a digital 12-bit word written by the Neuron, and is double-
// buffered in front of the position command DAC.  The position loop
// is closed in analog hardware.  The position command can be updated
// in one of two modes: in sync mode, the Neuron writes to the DAC
// buffer, and the DAC receives the updated command when a ZSYNC
// pulse is recieved from the page scanner controller (P/N 000134);
// in async mode, the Neuron writes to the DAC buffer and then forces
// the command into the DAC.
//    For autofocus purposes, there is a latching video comparator on
// the board, with a programmable threshold (similar to that on the
// PMT Amplifier (P/N 000276).  There is also a fast 8-bit ADC for use
// by autofocus routines involving the stepper motor Z-axis and the
// fast (piezo) Z-axis.  This same ADC can be used to read the high
// voltage driving the piezo crystal, and to read back the analog
// position feedback.
//    Finally, an emergency retract input is available that forces the
// position command to zero.  This feature is not used at present.
//
//    There is a custom bus structure on the FastZ PCB that allows
// Neuron to address many devices with an 8-bit bi-directional bus,
// yet uses only the general purpose I/O lines of the device. This
// allows a general purpose Neuron module to be used because the main
// data/address buses of the Neuron are not required. In this scheme,
// I/O bits 0-7 are used as the data bus, bit 8 is a read/not-write
// line, bit 9 is an address/not-data line, and bit 10 is a low-true
// strobe line. In general, the Neuron will write an address to the
// address register, then read or write the intended data value; the
// strobe line executes the intended operation after address and data
// have been appropriately set up.
//    The following rules are adhered to by the hardware and should
// be respected by software:
//    1. No one drives the data bus unless STB* (low-true STroBe) is
//       asserted. The only exception is that the Neuron drives the
//       bus with an address value without asserting STB* when
//       ADDR/DATA* is set high.
//    2. Setting the ADDR/DATA* (ADDRess/not-DATA) line forces a
//       write operation (the RD/WR* line is don't care when
//       ADDR/DATA* = 1).
//    3. The address latch is transparent, and controlled by ADDR/DATA*,
//       so no strobe is required to write the address.
//    4. Because of rule 1, no one should read the bus until STB* is
//       low.
// This bus arrangement means a lot of bit twiddling to do a read or
// write, but if the address does not change between operations, it
// can be speeded up considerably. FastRead and FastWrite routines are
//  provided that do not set up addresses are provided for this purpose.

#include <control.h>
```

```
//**********************************************************************
//********************** DEFINITIONS *****************************
//**********************************************************************
// Constants

// Mode Commands:
#define SELF_TEST           0x00        // commands from host
#define AUTO_FOCUS          0x10
#define READ_VIDEO          0x11
#define READ_FDBK           0x12
#define READ_HV             0x13
#define SET_FOR_VOL         0x14
#define MONITOR_FOCUS       0x15
#define ABORT_CMD           0x16
#define RD_STATUS           0x17
#define AF_FULLRANGE        0x18        // af function from 0 to 4095 tvt
#define AF_PMTVALS          0x19        // new af with pmt reading storage  tvt
#define AF_FAST_T           0x1A        // fast af algorithm for testing tvt
#define AF_HALFRANGE        0X1B        // af function from 1024 to 3036 tvt
#define AFFR_FAST           0X1C        // fast af function from 0 to4095 tvt
#define AF_FAST             0X1D        // fast af algorithm tvt
#define NULL_MODE           0xFF        // used to prevent repeated mode cmds

// status codes
#define SUCCESS             0
// autofocus responses:
#define NO_FOCUS_FF         1           // focus FF never tripped
#define TOO_FAR             2           // never saw increasing signal
#define NOT_FAR_ENUF        3           // never saw decreasing signal
#define END_OF_TRAVEL       4           // success, but at end of travel
// volume mode responses:
#define RDY_FOR_VOL         5
#define VOLUME_COMPL        6
#define UNDERFLOW           7
```

```
// coarse autofocus responses:
#define COMPARATOR_READY      8
#define THRESH_TOO_LOW        9
#define COMP_TRIPPED          0x0A
#define ABORTED               0x0B
// misc
#define CMD_UNKNOWN           0xFF

// Register addresses:  note that two MSBs of address are mux
// select bits.
// Write-Only addresses:
#define CTRL_REG          0          // Control register address
#define THRESH_DAC        1          // Autofocus threshold DAC
#define POS_CMD_LO        2          // Position command lo byte
#define POS_CMD_HI        3          // Position command hi byte
// Read-Only addresses:
#define STS_REG           0          // Status Register
#define ADC_ADDR          1          // ADC

// mux addresses: logical OR with register addresses
#define SEL_VIDEO         0          // video for autofocus
#define SEL_FDBK          0x40       // analog position feedback
#define SEL_HV            0x80       // high voltage output

// An 'n' suffix on a signal name indicates low true
// control register bits:
#define SYNC_MODE         0x01   // set to one for sync DAC update
#define ASYNC_TRIG        0x02   // pulse hi to update DAC in async mode
#define RST_ZSYNCn        0x04   // pulse low to reset ZSYNC flip flop
#define RST_FOCUSn        0x08   // pulse low to reset FOCUS flip flop
#define RST_RETRACTn      0x10   // pulse low to reset EMERG RETRACT
#define RST_INTEGn        0x20   // hold low to turn off position loop
                                 // integrator
#define SPARE_TEST        0x40
```

RECTIFIED SHEET (RULE 91)

```
#define TEST_LEDn           0x80   // set low to illuminate TEST LED

// status register bits:
#define ZSYNCDn             0x01   // low means ZSYNC has occurred
#define FOCUSDn             0x02   // low means FOCUS has occurred
#define RETRACTDn           0x04   // low means EMERG RETRACT has occurred
#define ADC_BUSYn           0x08   // low means conversion in process
#define TEST_IN             0x20
#define LED_TST_IN          0x40
#define TEST_JMPRn          0x80   // low means test jumper is in place

// control line defintions:
#define READ                1      // for RD_WRn line
#define WRITE               0
#define ADDR                1          // for ADDR_DATAn line
#define DATA                0

// autofocus constants:
#define MAX_Z               4095   // max position DAC range
#define AF_FAST_DELTA       10     // step size for fast autofocus sweep
#define AF_SLOW_DELTA       1      // step size for final autofocus
#define Z_INSURANCE         300    // distant to step down before fine AF
#define MIN_DELTA           15     // min delta from ADC to indicate slopes

// intial values
#define ZCTRL_INIT          0x22;
#define INIT_THRESH         0x16;


//**********************************************************************
//*********************** PRAGMAS *************************************
//**********************************************************************
#pragma net_buf_in_count  7
#pragma net_buf_out_count 7
```

```
//**********************************************************************
//********************** I/O DECLARATIONS *************************
//**********************************************************************
IO_0  output   byte    WRITE_DATA;          // parallel port
IO_0  input    byte    READ_DATA;

IO_8   output  bit     RD_WRn;              // ReaD/not-WRite
IO_9   output  bit     ADDR_DATAn;          // ADDRess/not-DATA
IO_10  output  bit     STBn;                // low-true STroBe


//**********************************************************************
//********************** NETWORK VARIABLES ************************
//**********************************************************************
network  output   polled unsigned SW_Rev = SWR;    // software version
network  output   unsigned   ResetSts;         // for indicating reset
network  input    unsigned   OpMode=0;         // for remote proc. calls
network  output   unsigned   OpStatus;         // response to calls
network  input    unsigned   OpParArray[2];  // Operation parameters
network  output   unsigned   OpStsArray[2];  // Extra Status info

network  input    unsigned AFocusThresh=16;        // 0-255 for Autofocus threshold
network  input    unsigned long ZPosCmd=0;         // Position Command 0-4095
network  input    unsigned Z_Control;        // Control byte
network  input    unsigned MuxSel=0;         // ADC mux selector
network  input    unsigned ZStep=4;          // step size for vol acquire
network  input    unsigned NumFrames=64;  // # frames for volume

network output polled unsigned Z_Status;              // Status byte
network output polled unsigned long PosStatus;        // more status

struct PMTARRAY            // define structure to hold pmt values   tvt
{
    unsigned    pmt[16];
  };

  // declare 16 network variables as structure of PMTARRAY type
  // to hold 256 pmt readings (one every 16 steps of the fast z stage tvt

  network output polled struct PMTARRAY pmtvals1;
  network output polled struct PMTARRAY pmtvals2;
  network output polled struct PMTARRAY pmtvals3;
  network output polled struct PMTARRAY pmtvals4;
  network output polled struct PMTARRAY pmtvals5;
  network output polled struct PMTARRAY pmtvals6;
  network output polled struct PMTARRAY pmtvals7;
  network output polled struct PMTARRAY pmtvals8;
  network output polled struct PMTARRAY pmtvals9;
  network output polled struct PMTARRAY pmtvals10;
  network output polled struct PMTARRAY pmtvals11;
  network output polled struct PMTARRAY pmtvals12;
  far network output polled struct PMTARRAY pmtvals13;
  far network output polled struct PMTARRAY pmtvals14;
  far network output polled struct PMTARRAY pmtvals15;
  far network output polled struct PMTARRAY pmtvals16;

  //**********************************************************************
  //********************** GLOBAL VARIABLES ***************************
  //**********************************************************************


  //**********************************************************************
  //********************** FUNCTION DEFINITIONS **********************
  //**********************************************************************
  unsigned short ReadPort (unsigned short TargetAddr)
  // read a port, including set up address
    {
```

```
    unsigned short result;

    io_set_direction (WRITE_DATA, IO_DIR_OUT);
    io_out (ADDR_DATAn, ADDR);                    // set ADDR_DATAn to ADDR
    io_out (WRITE_DATA, TargetAddr | MuxSel);     // write address
    io_out (ADDR_DATAn, DATA);                    // set ADDR_DATAn to DATA
    io_set_direction (READ_DATA, IO_DIR_IN);
    io_out (RD_WRn, READ);                        // set RD_WRn for READ
    io_out (STBn, 0);                             // assert STBn
    result = io_in (READ_DATA);                   // read data
    io_out (STBn, 1);                             // de-assert STBn
    return (result);                              // return result
    }


void WritePort (unsigned short TargetAddr, unsigned short WriteData)
// write to a port, including set up address
    {

    io_set_direction (WRITE_DATA, IO_DIR_OUT);
    io_out (ADDR_DATAn, ADDR);                    // set ADDR_DATAn to ADDR
    io_out (WRITE_DATA, TargetAddr | MuxSel);     // write address
    io_out (ADDR_DATAn, DATA);                    // set ADDR_DATAn to DATA
    // no need to change port direction for write
    io_out (RD_WRn, WRITE);                       // set RD_WRn for WRITE
    io_out (WRITE_DATA, WriteData);               // set up data
    io_out (STBn, 0);                             // assert STBn
    io_out (STBn, 1);                             // de-assert STBn
    }


unsigned FastRead (void)
// read a port, without address set up
    {
```

```
  unsigned result;

  io_out (STBn, 0);                        // assert STBn
  result = io_in (READ_DATA);              // read data
  io_out (STBn, 1);                        // de-assert STBn
  return (result);                         // return result
  }


void FastWrite (unsigned short WriteData)
// write to a port, without address set up
  {

  io_out (WRITE_DATA, WriteData);          // set up data
  io_out (STBn, 0);                        // assert STBn
  io_out (STBn, 1);                        // de-assert STBn
  }


void SetAddressRead (unsigned short TargetAddr)
// Setup address for fast read
  {

  io_set_direction (WRITE_DATA, IO_DIR_OUT);
  io_out (ADDR_DATAn, ADDR);                       // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, TargetAddr | MuxSel);        // write address
  io_out (ADDR_DATAn, DATA);                       // set ADDR_DATAn to DATA
  io_set_direction (READ_DATA, IO_DIR_IN);
  io_out (RD_WRn, READ);                           // set RD_WRn for READ
  }


void SetAddressWrite (unsigned short TargetAddr)
// Setup address for fast read
```

```
    {
    io_set_direction (WRITE_DATA, IO_DIR_OUT);
    io_out (ADDR_DATAn, ADDR);                     // set ADDR_DATAn to ADDR
    io_out (WRITE_DATA, TargetAddr | MuxSel);      // write address
    io_out (ADDR_DATAn, DATA);                     // set ADDR_DATAn to DATA
    // no need to change port direction for write
    io_out (RD_WRn, WRITE);                        // set RD_WRn for WRITE
    }


void SetCrtl (unsigned NewCtrl)
// sets the control word
    {
    Z_Control = NewCtrl;
    WritePort (CTRL_REG, NewCtrl);
    }


void MoveZ (unsigned long target)
// move the Z-axis to the current commanded position (ZPosCmd)
    {
    if (target > MAX_Z)
      target = MAX_Z;
    ZPosCmd = target;
    WritePort (POS_CMD_LO, (unsigned int)(ZPosCmd & 0xFF)); // set position
    WritePort (POS_CMD_HI, (unsigned int)(ZPosCmd >>8) & 0xFF);
    SetCtrl (Z_Control & (~ASYNC_TRIG) & (~SYNC_MODE)); // clock DAC
    SetCtrl (Z_Control | ASYNC_TRIG);
    }
```

ORIGINAL ALGORITHM

```
unsigned AFocusServo (unsigned Multiplier)
// Parameter is a multiplier for the speed of autofocus, and should
// be changed as a function of the numerical aperature of the
// objective, or as a function of the depth of focus. The depth of
// focus is inversely proportional to the square of the numerical
// aperature. For an NA of 1, the multiplier is 1, for a NA of
// 0.5, the multiplier should be 1/((0.5)^2) = 4, etc.
//
// The autofocus routines use a latching focus comparator with a
// programmable threshold (the 'focus flip-flop'), and a fast 8-bit
// ADC. Both of these devices monitor the video signal from the
// PMT preamplifier, but the focus flip flop uses a high-bandwidth
// video signal, and the ADC uses a low bandwidth version. The
// reason for this is that the focus flip flop must detect the
// focus condition, no matter how short in time it persists, but
// the ADC is used for the final autofocus operations and should
// deliver an average value of the video signal, representing the
// video level over a spacial 'patch' of the target image. Since the
// ZSYNC signal is also present on the FastZ board, it is possible
// to sample the ADC synchronous with the video frame and thereby
// obtain a repeatable autofocus. There is no attempt at this time
// to do that - we merely rely on the bandwith of the video input
// to the ADC being matched to the video frame rate. Of course, the
// video frame rate is adjustable, but the ADC bandwidth is not, so
// an approximation is the best we can do.
//
// Autofocus for the Fast Z-axis works as follows:
//   1. The stage is moved to its extreme lower limit (ZPosCmd = 0).
//   2. The autofocus threshold is appropriately set. 1/16 of full
//      scale might be a good starting value; we expect the autozero
//      routine to set the 'black' level to approx. 1/64 of full
//      scale. Note that the PMT gain must be appropriately set
//      for the incident laser power  (and wavelength) and type of
//      substrate being scanned.  An autogain cycle is expected after
//      autofocus completes.
//   3. The focus flip flop is cleared.
//   4. The stage is moved up a step at a time (but rapidly) while
//      the focus flip flop is monitored.
//   5. When the focus flip flop trips, the stage is stopped.  We expect
//      that stage may have to move up further to attain peak focus,
//      but depending on many factors (e.g. servo bandwidth, stage speed,
//      the rate at which the focus FF is sampled), the stage could
//      have moved beyond focus (this is almosta certain for the
//      stepper motor autofocus).
//   6. The stage is therefore moved back down a fixed amount to
//      ensure it is below the focus position.
//   7. The stage is then moved up slowly while the video signal is
//      monitored via the ADC.
//   8. When a local peak is detected above a second (software)
//      threshold, the movement stops and the stage returns to the peak.
//   9. If a peak is found, and the stage ends up at a point other
//      than the end points, the autofocus operation is considered
//      successful.  The final commanded position and success/failure
//      are status variables.
{
unsigned short BaseVal, LastVal, NextVal, BestVal;   // ADC values
unsigned long BestPos;                 // position of peak signal
unsigned long Move_Delta;
short Delta;
boolean PkFound;
unsigned Stat;

if (Multiplier == 0)
   Multiplier = 1;
SetCtrl (Z_Control & (~SYNC_MODE));   // async mode
// move to the extreme lower limit
MoveZ (0);
```

```
delay(1977);                              // wait 50 ms
// clear the focus FF
SetCtrl (Z_Control & (~RST_FOCUSn));
SetCtrl (Z_Control | RST_FOCUSn);
Move_Delta = AF_FAST_DELTA * Multiplier;
// we assume the autofocus flip flop threshold has been set
if ((ReadPort (STS_REG) & FOCUSDn) != 0)
  {                 // if focus FF not tripped immediatley
  // note that this loop should look at RETRACTDn, if it were used
  while (((ReadPort (STS_REG) & FOCUSDn) != 0) &&
       ((ZPosCmd + Move_Delta) < MAX_Z))
    {
    MoveZ (ZPosCmd + Move_Delta);
    watchdog_update();
    }
  if ((ZPosCmd + Move_Delta) < MAX_Z) // if focus FF tripped
    {
    if (ZPosCmd - Z_INSURANCE >= 0)
      MoveZ (ZPosCmd - Z_INSURANCE);        // back off
    else
      MoveZ (0);
    delay (4000);                 // wait 100 ms
    // now move up looking for peak signal
    PkFound = FALSE;
    MuxSel = SEL_VIDEO;
    NextVal = ReadPort (ADC_ADDR);
    LastVal = NextVal;
    BaseVal = NextVal;
    BestVal = NextVal;
    BestPos = ZPosCmd;
    Move_Delta = AF_SLOW_DELTA * Multiplier;
    // start by moving up until get an increasing signal
    while ((PkFound==FALSE) && ((ZPosCmd + Move_Delta)<MAX_Z))
      {
```

```
      MoveZ (ZPosCmd + Move_Delta);
      LastVal = NextVal;
      NextVal = ReadPort (ADC_ADDR);
      if (NextVal > BestVal)
        {
        BestPos = ZPosCmd;
        BestVal = NextVal;
        }
      Delta = NextVal - BaseVal;
      if (Delta > MIN_DELTA)
        PkFound = TRUE;
      watchdog_update();
      }
    if (PkFound == TRUE)        // if did find a rising slope
      {
      // now keep going until negative slope
      BaseVal = BestVal;
      PkFound = FALSE;
      while ((PkFound == FALSE) && ((ZPosCmd + Move_Delta) <
MAX_Z))
        {
        MoveZ (ZPosCmd + Move_Delta);
        LastVal = NextVal;
        NextVal = ReadPort (ADC_ADDR);
        if (NextVal > BestVal)
          {
          BestVal = NextVal;
          BaseVal = NextVal;
          BestPos = ZPosCmd;
          }
        Delta = BaseVal - NextVal;
        if (Delta > MIN_DELTA)
          PkFound = TRUE;
        watchdog_update();
```

```
                }
              if (PkFound == TRUE)
                {
                if ((ZPosCmd + Move_Delta) < MAX_Z)
                  Stat = SUCCESS;
                else
                  Stat = END_OF_TRAVEL;
                }
              else
                Stat = NOT_FAR_ENUF;
              }
            else
              Stat = TOO_FAR;
            MoveZ (BestPos);
            }
          else
            Stat = NO_FOCUS_FF;
          }
        else
          Stat = THRESH_TOO_LOW;
        return (Stat);
        }

      unsigned long AFFullRange(void)    // tvt
      // Function to step through entire fast z stage range at 16 counts per step
      // Stage is returned to fast z position which gives the highest PMT value.
      {
        unsigned short NextVal, BestVal;    // ADC values
        unsigned long BestPos;              // position of peak signal
        unsigned long step;                 // z stage step loop counter
        static int FocOff;                  // Focus offset

        // Set FocOff
        FocOff= OpParArray[0];
```

RECTIFIED SHEET (RULE 91)

```
    SetCtrl (Z_Control & (~SYNC_MODE));  // async mode
    // move to the extreme lower limit
    MoveZ (0);                           // Start at the bottom
    delay(1977);                         // wait 50 ms

// step through entire Z range checking for peak PMT and storing values

    NextVal=ReadPort(ADC_ADDR);          // Store current PMT value
    BestVal=NextVal;                     // Clear Largest value
    BestPos=0;                           // Clear Best Position

    for (step=0; step < = 4096; step + =16)  // Entire range in steps of 16
    {
       if(step = = 0) step = 1;
       MoveZ(step-1);                    // Move 0 to 4095
       watchdog_update();
       NextVal = ReadPort(ADC_ADDR);     // Store current PMT value

       if (NextVal > BestVal)            // If Current value is larger
       {                                 // than the best (largest) value
          BestPos=step;                  // this is the best position
          BestVal=NextVal;               // Update best value
       }
    }

    OpStatus = BestVal;                  // send the largest value out
    MoveZ(BestPos-1-FocOff);             // move to the best position
    return BestPos-1-FocOff;             // return the best positon
}
```

NEW ALGORITHM

```
unsigned long AFFast(void)    // tvt
// Fast Autofocus Function: This funciton does a coarse then a fine autofocus
//
```

```
{
  unsigned short NextVal, BestVal;    // ADC values
  unsigned long BestPos;              // position of peak signal
  unsigned long AvgVal;               // average value storage
  unsigned long FineCenter;           // center of fine focus range
  unsigned long Move_Delta;           // steps per moove
  unsigned long step;                 // z stage step loop counter
  unsigned m;                         // array member
  unsigned l;                         // loop counter
  static int FocOff;                  // Focus offset
  unsigned int StepSize;              // step size during coarse focus

  // Set FocOff and StepSize
      FocOff= OpParArray[0];
      StepSize = 128;

  SetCtrl (Z_Control & (~SYNC_MODE));  // async mode
  // move to the extreme lower limit
  MoveZ (0);                            // Start at the bottom
  delay(1977);                          // wait 50 ms

// ***** coarse autofocus *****
// step through entire Z range with large steps
// checking for peak PMT and storing values
  AvgVal=0;
  MuxSel = SEL_VIDEO;
  AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
  AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
  AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
  AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
  NextVal=AvgVal/4;
  BestVal=NextVal;                      // Clear Largest value
  BestPos=0;                            // Clear Best Position
```

```
    for (step=0; step < = 4096; step + =StepSize)  // Entire range in steps of 1
    {
        if(step = = 0) step = 1;
        MoveZ(step-1);                              // Move 0 to 4095
        watchdog_update();
        AvgVal=0;
        MuxSel = SEL_VIDEO;
        AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
        AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
        AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
        AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
        NextVal=AvgVal/4;

        if (NextVal > BestVal)                 // If Current value is larger
        {                                      // than the best (largest) value
            BestPos=step;                      // this is the best position
            BestVal=NextVal;                   // Update best value
        }
    }

// Move to start value for fine autofocus and wait
    if (BestPos > 129)
        MoveZ (BestPos-128-1); // Start at the bottom
    else
        MoveZ (1);
    delay(395);                      // wait 10 ms

// Set up comparison values
    AvgVal=0;
    MuxSel = SEL_VIDEO;
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
```

```
        NextVal=AvgVal/4;
        BestVal=NextVal;                          // Clear Largest value

    // ***** fine autofocus *****
    // using position from coarse focus
        FineCenter=BestPos; // Set center of focus to the best coarse focus pos.
        if(FineCenter<129)  // Make sure it's valid (at least 129)
            FineCenter=129;
        for (step = FineCenter-128; step <= FineCenter+128; step +=8)  // Entire r
        {
            MoveZ(step-1);                         // Move through fine af range
            watchdog_update();
            AvgVal=0;
            MuxSel = SEL_VIDEO;
            AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
            AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
            AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
            AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
            NextVal=AvgVal/4;

            if (NextVal > BestVal)          // If Current value is larger
            {                               // than the best (largest) value
                BestPos=step;               // this is the best position
                BestVal=NextVal;            // Update best value
            }
        }

        OpStatus = BestVal;                 // send the largest value out
        MoveZ(BestPos-1-FocOff);            // move to the best position
        return BestPos-1-FocOff;            // return the best positon
    }

unsigned long AFFastTest(void)    // tvt
// Fast Autofocus Function: This funciton does a coarse then a fine autofocus.
// It is a test version of AFFast w/ variable averaging and step size and
// recording of PMT Values in network variables.
{                                                  PROFILE ALGORITHM
  unsigned short NextVal, BestVal;    // ADC values
  unsigned long  AvgVal;              // Average PMT Value
  unsigned long BestPos;              // position of peak signal
  unsigned long FineCenter;           // center of fine focus range
  unsigned long Move_Delta;           // steps per moove
  unsigned long step;                 // z stage step loop counter
  unsigned m;                         // array member
  unsigned l;                         // loop counter
  unsigned a;                         // averaging setting
  static int FocOff;                  // Focus offset
  unsigned int StepSize;              // step size during coarse focus

  // Set FocOff and StepSize
     a = OpParArray[0];
     StepSize = OpParArray[1];

// Make sure StepSize is good (>7)
   if (StepSize < 8)     // must be at least 8
      StepSize = 8;

// Make sure averaging is good (>3)
   if (a < 4)            // must be at least 4
      a = 4;

  SetCtrl (Z_Control & (~SYNC_MODE));  // async mode
  // move to the extreme lower limit
  MoveZ (0);                           // Start at the bottom
  delay(1977);                         // wait 50 ms

// ***** coarse autofocus *****
// step through entire Z range with large steps
// checking for peak PMT and storing values
    AvgVal=0;
    for (l=0;l<a;l++)
        {
        MuxSel = SEL_VIDEO;
        AvgVal=AvgVal+ReadPort(ADC_ADDR);       // Store current PMT value
        }
    NextVal=AvgVal/a;
    BestVal=NextVal;                    // Clear Largest value
    BestPos=0;                          // Clear Best Position

    for (step=0; step <= 4096; step +=StepSize)   // Entire range in steps of 1
    {
        if(step == 0) step = 1;
        MoveZ(step-1);                  // Move 0 to 4095
        watchdog_update();
        AvgVal=0;
        for(l=0;l<a;l++)
        {
            MuxSel = SEL_VIDEO;
            AvgVal = AvgVal+ReadPort(ADC_ADDR);       // Get current PMT value
        }
        NextVal=AvgVal/a;


            if (NextVal > BestVal)              // If Current value is larger
            {                                   // than the best (largest) value
                BestPos=step;                   // this is the best position
                BestVal=NextVal;                // Update best value
        }

        /* Write PMT Values to nv's */

        if (StepSize == 128)
```

```
            {
               if (step < 2048)  // use structure 1
               {
                  m = step/StepSize;
                  pmtvals1.pmt[m] = NextVal;
               }

               else if ((step >= 2048) && (step < 4096)) // use structure 2
               {
                  m = (step-2048)/StepSize;
                  pmtvals2.pmt[m]=NextVal;
               }
            }

         }

   // Move to start value for fine autofocus and wait
      if (BestPos > 129)
         MoveZ (BestPos-128-1); // Start at the bottom
      else
         MoveZ (1);
      delay(395);                    // wait 10 ms

   // Set up comparison values
         AvgVal=0;
         for(l=0;l<a;l++)
         {
            MuxSel = SEL_VIDEO;
            AvgVal = AvgVal+ReadPort(ADC_ADDR);      // Get current PMT value
         }
         NextVal=AvgVal/a;
         BestVal=NextVal;                    // Clear Largest value
```

```
// ***** fine autofocus *****
// using position from coarse focus
   FineCenter=BestPos; // Set center of focus to the best coarse focus pos.
   if(FineCenter<129)    // Make sure it's valid (at least 129
      FineCenter=129;
   for (step = FineCenter-128; step <= FineCenter+128; step +=8)
// go through fine autofocus range in steps of 8
   {
      MoveZ(step-1);                   // Move through fine af range
      watchdog_update();
      AvgVal=0;
      for(l=0;l<a;l++)
      {
         MuxSel = SEL_VIDEO;
         AvgVal = AvgVal+ReadPort(ADC_ADDR);      // Get current PMT value
      }
      NextVal=AvgVal/a;

      if (NextVal > BestVal)           // If Current value is larger
      {                                // than the best (largest) value
         BestPos=step;                 // this is the best position
         BestVal=NextVal;              // Update best value
      }

      /* Write PMT Values to nv's */

         if (step < FineCenter) // use structure 3
         {
            m = 16-((FineCenter-step)/8);
            pmtvals3.pmt[m] = NextVal;
         }

         else if (step >= FineCenter) // use structure 4
         {
                m = (step-FineCenter)/8;
                pmtvals4.pmt[m]=NextVal;
            }

    }

    OpStatus = BestVal;             // send the largest value out
    MoveZ(BestPos-1-FocOff);        // move to the best position
    return BestPos-1-FocOff;        // return the best positon
}
```

OTHER FUNCTIONS

```
unsigned long AFPMTVals(void)    // tvt
// Function to step through entire fast z stage range at 16 counts per step
// storing PMT values at each step in network variable structures (16 structure
// of 16 element arrays).  Stage is returned to fast z position which gives the
// highest PMT value.
{
   unsigned short NextVal, BestVal;    // ADC values
   unsigned long BestPos;              // position of peak signal
   unsigned long Move_Delta;           // steps per moove
   unsigned long step;                 // z stage step loop counter
   unsigned m;                         // array member
   static int FocOff;                  // Focus offset
//   short Delta;
//   unsigned Stat;

   SetCtrl (Z_Control & (~SYNC_MODE));   // async mode
   // move to the extreme lower limit
   MoveZ (0);
   delay(1977);                     // wait 50 ms

// step through Z range checking for peak PMT and storing values
```

```
NextVal=ReadPort(ADC_ADDR);
BestVal=NextVal;
BestPos=0;
************************
for (step=0; step <= 4096; step +=16)
{
   if(step == 0) step = 1;
   MoveZ(step-1);
   watchdog_update();
   NextVal = ReadPort(ADC_ADDR);

   if (NextVal > BestVal)
   {
      BestPos=step;
      BestVal=NextVal;
   }

   /* Write PMT Values to nv's */

   if (step < 256)  // use structure 1
   {
      m = step/16;
      pmtvals1.pmt[m] = NextVal;
   }

   else if ((step >= 256) && (step < 512)) // use structure 2
   {
      m = (step-256)/16;
      pmtvals2.pmt[m]=NextVal;
   }

   else if ((step >= 512) && (step < 768)) // use structure 3
   {
      m= (step-512)/16;
```

```
    pmtvals3.pmt[m]=NextVal;
}

else if ((step >= 768) && (step < 1024)) // use structure 4
{
    m = (step - 768) / 16;
    pmtvals4.pmt[m]=NextVal;
}

else if ((step >= 1024) && (step < 1280)) // use structure 5
{
    m = (step - 1024) / 16;
    pmtvals5.pmt[m]=NextVal;
}

else if ((step >= 1280) && (step < 1536)) // use structure 6
{
    m = (step - 1280) / 16;
    pmtvals6.pmt[m]=NextVal;
}

else if ((step >= 1536) && (step < 1792)) // use structure 7
{
    m = (step - 1536) / 16;
    pmtvals7.pmt[m]=NextVal;
}

else if ((step >= 1793) && (step < 2048)) // use structure 8
{
    m = (step - 1793) / 16;
    pmtvals8.pmt[m]=NextVal;
}

else if ((step >= 2048) && (step < 2304)) // use structure 9
```

```
{
   m = (step - 2048) / 16;
   pmtvals9.pmt[m]=NextVal;
}

else if ((step > = 2304) && (step < 2560)) // use structure 10
{
   m = (step - 2304) / 16;
   pmtvals10.pmt[m]=NextVal;
}

else if ((step > = 2560) && (step < 2816)) // use structure 11
{
   m = (step - 2560) / 16;
   pmtvals11.pmt[m]=NextVal;
}

else if ((step > = 2816) && (step < 3072)) // use structure 12
{
   m = (step - 2816) / 16;
   pmtvals12.pmt[m]=NextVal;
}

else if ((step > = 3072) && (step < 3328)) // use structure 13
{
   m = (step - 3072) / 16;
   pmtvals13.pmt[m]=NextVal;
}

else if ((step > = 3328) && (step < 3584)) // use structure 14
{
   m = (step - 3328) / 16;
   pmtvals14.pmt[m]=NextVal;
}
```

```
            else if ((step >= 3584) && (step < 3840)) // use structure 15
            {
               m = (step - 3584) / 16;
               pmtvals15.pmt[m]=NextVal;
            }

            else // if ((step >= 3840) && (step < 4096)) // use structure 16
            {
               m = (step - 3840) / 16;
               pmtvals16.pmt[m]=NextVal;
            }



         }

         OpStatus = BestVal;               // send the largest value out
         MoveZ(BestPos-1-FocOff);          // move to the best position
         return BestPos-1-FocOff;          // return the best positon
      }


      unsigned CheckParms(void)
        {
        signed long FinalPos;
        unsigned short RetVal;

        FinalPos = ZPosCmd - (NumFrames-1)*ZStep;
        if (FinalPos < 0)
          RetVal = UNDERFLOW;
        else
          RetVal = RDY_FOR_VOL;
        return (RetVal);
        }
```

```
unsigned SelfTest (void)
  {
  unsigned AllErrors, OldCtrl;

  AllErrors = 0;
  OldCtrl = Z_Control;
  // turn off two test bits
  SetCtrl ( Z_Control & (~(SPARE_TEST | TEST_LEDn)));
  if ((ReadPort (STS_REG) & (TEST_IN | LED_TST_IN)) != 0)
    AllErrors = AllErrors | 0x1;
  SetCtrl ( Z_Control | SPARE_TEST);
  if ((ReadPort (STS_REG) & TEST_IN) != TEST_IN)
    AllErrors = AllErrors | 0x2;
  SetCtrl ( (Z_Control | TEST_LEDn) & (~SPARE_TEST));
  if ((ReadPort (STS_REG) & (TEST_IN | LED_TST_IN)) != LED_TST_IN)
    AllErrors = AllErrors | 0x4;

  SetCtrl (OldCtrl);
  return (AllErrors);
  }



//************************************************************************
//************************* WHEN STATEMENTS ***********************
//************************************************************************
when (reset)
  {
  unsigned long i;

  ResetSts = 0;                          // indicate reset to host
  Z_Control = ZCTRL_INIT;
  WritePort (CTRL_REG, Z_Control);
  AFocusThresh = INIT_THRESH;
```

```
MoveZ (0);
MuxSel = SEL_VIDEO;

if (((Z_Status = ReadPort (STS_REG)) & TEST_JMPRn) == 0) // if test mode
  {                                              // exercise everything
  SetCtrl (Z_Control);
  while (TRUE)                                   // until next reset
    {
    for (i=0; i<4096; i+=512)
      {
      MoveZ (i);
      ReadPort (STS_REG);
      delay(787);
      watchdog_update();
      }

     for (i=4095; i>=512; i-=512)
      {
      MoveZ (i);
      ReadPort (STS_REG);
      delay(787);
      watchdog_update();
      }
    }
  }
else
  {
  SetCtrl (TEST_LEDn | ASYNC_TRIG);          // set control bits: async mode,
                                             // reset all FF's, reset integ,
                                             // turn off test LED
  WritePort (THRESH_DAC, AFocusThresh);
  SetCtrl (TEST_LEDn | RST_RETRACTn | RST_FOCUSn | RST_INTEGn
|
         RST_ZSYNCn | ASYNC_TRIG);    // release resets except integ
```

```
        for (i=ZPosCmd; i>1; i--)
          MoveZ (ZPosCmd);
        }
  /*
      unsigned long i;
      ResetSts = 0;              ?? indicate reset to host
      Z_Control = ZCTRL_INIT;
      WritePort (CTRL_REG, Z_Control);
      AFocusThresh = INIT_THRESH;
      MoveZ (0);
      MuxSel = SEL_VIDEO;

      if (((Z_Status = ReadPort (STS_REG)) & TEST_JMPRn) == 0) ?? if test mode
        {                                  ?? exercise everything
        SetCtrl (0);
        while (TRUE)                       ?? until next reset
          {
          watchdog_update();
          SetCtrl (~Z_Control);            ?? toggle ctrl bits
          if (Z_Control == 0)
            {
            SetCtrl (ASYNC_TRIG);          ?? write to DAC
            SetCtrl (0);
            }
          WritePort (THRESH_DAC, AFocusThresh);      ?? set threshold
          AFocusThresh++;
          ZPosCmd++;
          WritePort (POS_CMD_LO, (unsigned int)((ZPosCmd % 4) & 0xFF));
          WritePort (POS_CMD_HI, (unsigned int)((ZPosCmd % 4) >>8) & 0xFF);
          ReadPort (STS_REG);
          Z_Status = ReadPort (ADC_ADDR);
          post_events();
          }
        }
```

```
    else
      {
      SetCtrl (TEST_LEDn | ASYNC_TRIG);       ?? set control bits: async mode,
                                           ?? reset all FF's, reset integ,
                                           ?? turn off test LED
      WritePort (THRESH_DAC, AFocusThresh);
      SetCtrl (TEST_LEDn | RST_RETRACTn | RST_FOCUSn | RST_INTEGn
 |
              RST_ZSYNCn | ASYNC_TRIG);     ?? release resets except integ
      }
      OpStatus = 5;

  */

    }

  when (nv_update_occurs (AFocusThresh))
  // Update the auto focus threshold DAC
    {
    WritePort (THRESH_DAC, AFocusThresh);
    }


  when (nv_update_occurs (ZPosCmd))
  // update the position command DAC
    {
    if (ZPosCmd > MAX_Z)
      ZPosCmd = MAX_Z;
    MoveZ (ZPosCmd);
    }

  when (nv_update_occurs (Z_Control))
  // update the control word
    {
```

```
  WritePort (CTRL_REG. Z_Control);
  }

// ****************************************************************
// *************** remote procedure calls ****************************
// ****************************************************************
// when (nv_update_occurs) is not used here so that OpMode can
// be monitored within the when clauses, specifically to break
// out of the MONITOR_FOCUS mode.  OpMode is changed to NULL_MODE
// at the end of each clause to ensure that they are only invoked
// once.

when (OpMode == SELF_TEST)
  {
  OpStatus = SelfTest();
  OpMode = NULL_MODE;
  }

when (OpMode == AUTO_FOCUS)
  // Perform autofocus function)
  {
  OpStatus = AFocusServo (OpParArray[0]);
  PosStatus = ZPosCmd;
  OpMode = NULL_MODE;
  }

when (OpMode == AF_FULLRANGE)
  {
  // Perform full range autofocus function
        MuxSel = SEL_VIDEO;
        PosStatus = AFFullRange();
        OpMode = NULL_MODE;
     }
```

```
when (OpMode == AF_FAST)
   {
   // Perform fast autofocus function
         MuxSel = SEL_VIDEO;
         PosStatus = AFFast();
         OpMode = NULL_MODE;
      }


when (OpMode == AF_FAST_T)
   {
   // Perform fast autofocus function
         MuxSel = SEL_VIDEO;
         PosStatus = AFFastTest();
         OpMode = NULL_MODE;
      }

when (OpMode == AF_PMTVALS)
   {
   // Perform autofocus function which writes PMT values
         MuxSel = SEL_VIDEO;
         PosStatus = AFPMTVals();
         OpMode = NULL_MODE;
      }

when (OpMode == READ_VIDEO)
  // Read ADC on stated mux channel)
  {
  MuxSel = SEL_VIDEO;
  OpStatus = ReadPort (ADC_ADDR);
  OpMode = NULL_MODE;
      }

when (OpMode == READ_HV)
```

```
    // Read ADC on stated mux channel)
    {
    MuxSel = SEL_HV;
    OpStatus = ReadPort (ADC_ADDR);
    OpMode = NULL_MODE;
        }

when (OpMode == READ_FDBK)
    // Read ADC on stated mux channel)
    {
    MuxSel = SEL_FDBK;
    OpStatus = ReadPort (ADC_ADDR);
    OpMode = NULL_MODE;
    }

when (OpMode == SET_FOR_VOL)
    // setup for volume acquisition)
    {
    unsigned FrameCtr;

    if ((OpStatus = CheckParms()) == RDY_FOR_VOL)
        {
        // clear ZSYNC FF, wait for syncs and step when they come
        // assumes have moved to start position (closest to objective)
        // assumes ZSYNCs have been stopped, so clear ZSYNC FF and wait
        SetCtrl (Z_Control & (~RST_ZSYNCn));                    // clear ZSYNC
        SetCtrl (Z_Control | SYNC_MODE | RST_ZSYNCn);           // go to sync
mode
        // load DAC register with current position so it stays put
        // on first ZSYNC
        WritePort (POS_CMD_LO, (unsigned int)(ZPosCmd & 0xFF));
        WritePort (POS_CMD_HI, (unsigned int)(ZPosCmd >>8) & 0xFF);
        // alert host that ready for volume
        OpStatus = RDY_FOR_VOL;
```

```
          post_events();
          // take the volume)
          for (FrameCtr = 0; FrameCtr < NumFrames; FrameCtr++)
            {
            // wait for ZSYNC
            while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
              ;
            // reset the ZSYNC flip flop
            SetCtrl (Z_Control & (~RST_ZSYNCn));
            SetCtrl (Z_Control | RST_ZSYNCn);
            // step down
            ZPosCmd = ZPosCmd - ZStep;
            WritePort (POS_CMD_LO, (unsigned int)(ZPosCmd & 0xFF));
            WritePort (POS_CMD_HI, (unsigned int)(ZPosCmd >>8) & 0xFF);
            watchdog_update();          // prevent reset
            }
         SetCtrl (Z_Control & (~SYNC_MODE)); // async mode
         OpStatus = VOLUME_COMPL;
         }
      PosStatus = ZPosCmd;
      OpMode = NULL_MODE;
      }

    when (OpMode == MONITOR_FOCUS)
      // Clear focus flipflop and wait for trip
      {
      // we assume the autofocus flip flop threshold has been set
      // note that this loop should look at RETRACTDn, if it were used
      // clear the focus FF
      SetCtrl (Z_Control & (~RST_FOCUSn));
      SetCtrl (Z_Control | RST_FOCUSn);
      if ((ReadPort (STS_REG) & FOCUSDn) == 0)   // if trip immediately
        OpStatus = THRESH_TOO_LOW;
      else
    OpStatus = COMPARATOR_READY;
  post_events();
      // now hang out and wait for tripped
  while (((ReadPort (STS_REG) & FOCUSDn) != 0) &&
          (OpMode == MONITOR_FOCUS))
    {
    post_events();
    }
  if (OpMode == MONITOR_FOCUS)
    OpStatus = COMP_TRIPPED;
  else
    OpStatus = ABORTED;
  post_events();
  OpMode = NULL_MODE;
  }

when (OpMode == RD_STATUS)
  // read status reg
  {
  OpStatus = ReadPort (STS_REG);
  OpMode = NULL_MODE;
      }

when (OpMode == ABORT_CMD)
  // this shouldn't happen unless we are in another op mode
  {
  OpStatus = ABORTED;
  OpMode = NULL_MODE;
  }
```

```
unsigned ReadVidLoop(void)
{
    unsigned    l,a;            // loop and average counter
    unsigned    MaxVid;          // Maximum Video Value
    unsigned    CurVid;          // Current Video Value
    unsigned long AvgVid;        // Average Video
    int         MaxStatus;      // Satus of MaxVal
    unsigned    Avg;             // Averaging value
    unsigned    Count;           // loop count

// InMaximum to starting value

    OpStatus 0;                 // Clear OpStatus
    OpStsArray[0] 0;             // Clear OpParArray[0]
    OpStsArray[1] 0;             // Clear OpParArray[1]
    Count OpParArray[0];          // Set loop count
    Avg OpParArray[1];            // Set averaging

    if ((Avg<1) || (Avg>255))
        Avg 1;                  // Make sure averaging is in range

    if ((Count < 64) || (Count > 255))
        Count 64;               // Make sure count is in range


// PMT Read Loop
    for (l 0;l< Count;l++)
    // for 64 counts
    {
        AvgVid 0;

        for (a 0; a<Avg; a++)
        AvgVid  AvgVid + ReadPort(A C A R); // Get Current Video Level
```

```
CurVid  AvgVid/a;              // averaged over a

if (CurVid > MaxVid)           // If Current level is greater
{
   MaxVid CurVid;              // than max, reset max.
   MaxStatus 1;                // status of maximum.
}

// Store Video Level

if (l<ARRAYSIZE)
{
   pmtvals1.pmt[l%ARRAYSIZE] CurVid;      // Store Current Video
   if (MaxStatus  1)                      // if this is max
   {
      OpStsArray[0] 0;                    // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
   }
}
else if ((l> ARRAYSIZE) && (l<ARRAYSIZE*2))
{
   pmtvals2.pmt[l%ARRAYSIZE] CurVid;      // Store Current Video
   if (MaxStatus  1)                      // if this is max
   {
      OpStsArray[0] 1;                    // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
   }
}

else if ((l> ARRAYSIZE*2) && (l<ARRAYSIZE*3))
{
   pmtvals3.pmt[l%ARRAYSIZE] CurVid;      // Store Current Video
   if (MaxStatus  1)                      // if this is max
   {
```

```
            OpStsArray[0] 2;                  // Set Par0 to array
            OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*3) && (l<ARRAYSIZE*4))
    {
        pmtvals4.pmt[l%ARRAYSIZE] CurVid;     // Store Current Video
        if (MaxStatus  1)                     // if this is max
        {
            OpStsArray[0] 3;                  // Set Par0 to array
            OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*4) && (l<ARRAYSIZE*5))
    {
        pmtvals5.pmt[l%ARRAYSIZE] CurVid;     // Store Current Video
        if (MaxStatus  1)                     // if this is max
        {
            OpStsArray[0] 4;                  // Set Par0 to array
            OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*5) && (l<ARRAYSIZE*6))
    {
        pmtvals6.pmt[l%ARRAYSIZE] CurVid;     // Store Current Video
        if (MaxStatus  1)                     // if this is max
        {
            OpStsArray[0] 5;                  // Set Par0 to array
            OpStsArray[1] l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*6) && (l<ARRAYSIZE*7))
    {
        pmtvals7.pmt[l%ARRAYSIZE] CurVid;     // Store Current Video
        if (MaxStatus  1)                    // if this is max
        {
            OpStsArray[0] 6;                 // Set Par0 to array
            OpStsArray[1] 1%ARRAYSIZE;            // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*7) && (l<ARRAYSIZE*8))
    {
        pmtvals8.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
        if (MaxStatus  1)                    // if this is max
        {
            OpStsArray[0] 7;                 // Set Par0 to array
            OpStsArray[1] 1%ARRAYSIZE;            // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*8) && (l<ARRAYSIZE*9))
    {
        pmtvals9.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
        if (MaxStatus  1)                    // if this is max
        {
            OpStsArray[0] 8;                 // Set Par0 to array
            OpStsArray[1] 1%ARRAYSIZE;            // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*9) && (l<ARRAYSIZE*10))
    {
        pmtvals10.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
        if (MaxStatus  1)                    // if this is max
        {
            OpStsArray[0] 9;                 // Set Par0 to array
            OpStsArray[1] 1%ARRAYSIZE;            // Set Par1 to element
        }
    }
    else if ((l> ARRAYSIZE*10) && (l<ARRAYSIZE*11))
```

```
{
   pmtvals11.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
   if (MaxStatus  1)                     // if this is max
   {
      OpStsArray[0] 10;                  // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;         // Set Par1 to element
   }
}
else if ((l>  ARRAYSIZE*11) && (l<ARRAYSIZE*12))
{
   pmtvals12.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
   if (MaxStatus  1)                     // if this is max
   {
      OpStsArray[0] 11;                  // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;         // Set Par1 to element
   }
}
else if ((l>  ARRAYSIZE*12) && (l<ARRAYSIZE*13))
{
   pmtvals13.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
   if (MaxStatus  1)                     // if this is max
   {
      OpStsArray[0] 12;                  // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;         // Set Par1 to element
   }
}
else if ((l>  ARRAYSIZE*13) && (l<ARRAYSIZE*14))
{
   pmtvals14.pmt[l%ARRAYSIZE] CurVid;    // Store Current Video
   if (MaxStatus  1)                     // if this is max
   {
      OpStsArray[0] 13;                  // Set Par0 to array
      OpStsArray[1] l%ARRAYSIZE;         // Set Par1 to element
   }
```

```
        }
        else if ((l> ARRAYSIZE*14) && (l<ARRAYSIZE*15))
        {
           pmtvals15.pmt[l%ARRAYSIZE] CurVid;      // Store Current Video
           if (MaxStatus 1)                   // if this is max
           {
              OpStsArray[0] 14;                  // Set Par0 to array
              OpStsArray[1] 1%ARRAYSIZE;           // Set Par1 to element
           }
        }
        else // if ((l> ARRAYSIZE*15) && (l<ARRAYSIZE*16))
        {
           pmtvals16.pmt[l%ARRAYSIZE] CurVid;      // Store Current Video
           if (MaxStatus 1)                   // if this is max
           {
              OpStsArray[0] 15;                  // Set Par0 to array
              OpStsArray[1] 1%ARRAYSIZE;           // Set Par1 to element
           }
        }

        MaxStatus 0;
        watchdog update();
     }
  return MaxVid;
  }
  □
```

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX B

M-2464-1P US

```
// Fast Z-Axis Controller

// Source File:     Fastz.nc
// Author:          Chris F.
// Target Board:    0001003, Fast Z-Axis Controller
// Target Socket:   M1-U2
// Ultrapointe Corp.
// PROPRIETARY! ALL RIGHTS RESERVED
// **** FOR INTERNAL USE ONLY ****
//
// Revision History:
//
//

// !!!!****** REVISION NUMBER *******!!!!
#define SWR 0x21        // software revision 2.1
//
//
//

// The FastZ PCB controls a piezo-driven stage with analog position
// feedback.  The position command is a digital 14-bit word
// (12 bits used) written by the Neuron, and is double-
// buffered in front of the position command DAC.  The position loop
// is closed in analog hardware.  The position command can be updated
// in one of two modes: in sync mode, the Neuron writes to the DAC
// buffer, and the DAC receives the updated command when a ZSYNC
// pulse is recieved from the page scanner controller (P/N 000134);
// in async mode, the Neuron writes to the DAC buffer and the DAC is
// updated when hi byte is written.
//    For autofocus purposes, there is a latching video comparator on
// the board, with a programmable threshold (similar to that on the
// PMT Amplifier (P/N 000276).  There is also a fast 8-bit ADC for use
// by autofocus routines involving the stepper motor Z-axis and the
// fast (piezo) Z-axis.  This same ADC can be used to read the analog
// position feedback or a peak detector gated by ZSYNC.
//
//   There is a custom bus structure on the FastZ PCB that allows
// Neuron to address many devices with an 8-bit bi-directional bus,
// yet uses only the general purpose I/O lines of the device.  This
// allows a general purpose Neuron module to be used because the main
// data/address buses of the Neuron are not required.  In this scheme,
// I/O bits 0-7 are used as the data bus, bit 8 is a read/not-write
// line, bit 9 is an address/not-data line, and bit 10 is a low-true
// strobe line.  In general, the Neuron will write an address to the
// address register, then read or write the intended data value; the
// strobe line executes the intended operation after address and data
// have been appropriately set up.
//   The following rules are adhered to by the hardware and should
// be respected by software:
//   1. No one drives the data bus unless STB* (low-true STroBe) is
//      asserted.  The only exception is that the Neuron drives the
//      bus with an address value without asserting STB* when
//      ADDR/DATA* is set high.
//   2. Setting the ADDR/DATA* (ADDRess/not-DATA) line forces a
//      write operation (the RD/WR* line is don't care when
//      ADDR/DATA* = 1).
//   3. The address latch is transparent, and controlled by ADDR/DATA*,
//      so no strobe is required to write the address.
//   4. Because of rule 1, no one should read the bus until STB* is
//      low.
// This bus arrangement means a lot of bit twiddling to do a read or
// write, but if the address does not change between operations, it
// can be speeded up considerably.  FastRead and FastWrite routines are
//  provided that do not set up addresses are provided for this purpose.

#include <control.h>
```

```
//*****************************************************************
//********************** DEFINITIONS ******************************
//*****************************************************************
// Constants

// Mode Commands:
#define SELF_TEST     0x00    // commands from host
#define READ_VIDEO    0x11
#define READ_FDBK     0x12
#define READ_HV       0x13
#define SET_FOR_VOL   0x14
#define MONITOR_FOCUS 0x15
#define ABORT_CMD     0x16
#define RD_STATUS     0x17
#define AF_PMTVALS    0x19    // new af with pmt reading storage  tvt
#define AF_PEAK_SYNC  0x1C    // patch scan autofocus w/ peak detect & sync
#define AF_FAST       0X1D    // fast af algorithm tvt
#define RD_VID_LP     0X1E    // funcion to read PMT 64 times in 1 sec.
#define NULL_MODE     0xFF    // used to prevent repeated mode cmds

// status codes
#define SUCCESS       0

// volume mode responses:
#define RDY_FOR_VOL   5
#define VOLUME_COMPL  6
#define UNDERFLOW     7

// coarse autofocus responses:
#define COMPARATOR_READY 8
#define THRESH_TOO_LOW   9
#define COMP_TRIPPED     0x0A
#define ABORTED          0x0B
```

```
// misc
#define CMD_UNKNOWN  0xFF
#define DAC_SHIFT    2          // shift value for Z Command DAC

// Register addresses:  note that two MSBs of address are mux
// select bits.
// Write-Only addresses:
#define CTRL_REG      0       // Control register address
#define THRESH_DAC    1        // Autofocus threshold DAC
#define POS_CMD_LO    2        // Position command lo byte
#define POS_CMD_HI    3        // Position command hi byte

// Read-Only addresses:
#define STS_REG       0       // Status Register
#define ADC_ADDR      1        // ADC
#define RST_ZSYNCn      0x02    // pulse low to reset ZSYNC flip flop

// mux addresses: logical OR with register addresses
#define SEL_VIDEO     0       // video for autofocus
#define SEL_FDBK      0x40    // analog position feedback
#define SEL_HV        0x80    // high voltage output
#define SEL_PKDET     0x80    // peak detector replaces HV input

// An 'n' suffix on a signal name indicates low true
// control register bits:
#define SYNC_MODE    0x01       // set to one for sync DAC update
#define SPARE_TEST1  0x04       // control reg test line 1
#define RST_FOCUSn  0x08       // pulse low to reset FOCUS flip flop
#define SPARE_TEST2  0x10       // control reg test line 2
#define RST_INTEGn  0x20       // hold low to turn off position loop
                              // integrator
#define PKDET_MODE  0x40       // spare now allocated to peak detector AF
#define TEST_LEDn   0x80       // set low to illuminate TEST LED
```

```
// status register bits:
#define ZSYNCDn     0x01     // low means ZSYNC has occurred
#define FOCUSDn     0x02     // low means FOCUS has occurred
#define TEST_IN1    0x04     // first test line
#define ADC_BUSYn   0x08     // low means conversion in process
#define TEST_IN2    0x10     // second test line
#define PKDET_IN    0x20     // test peak det line
#define LED_TST_IN  0x40     // test led line
#define TEST_JMPRn  0x80     // low means test jumper is in place

// control line defintions:
#define READ        1        // for RD_WRn line
#define WRITE       0
#define ADDR        1              // for ADDR_DATAn line
#define DATA        0

// autofocus constants:
//* #define MAX_Z      4095     // max position DAC range
#define AF_FAST_DELTA  10     // step size for fast autofocus sweep
#define AF_SLOW_DELTA  1      // step size for final autofocus
#define ARRAYSIZE      16     // size of arrays for PMT values
#define COARSE_STEP    64     // step sizes for coarse and fine portions
#define FINE_STEP      8      // of autofocus
#define FINE_RANGE     128    // 1/2 of fine range
#define DAC_RANGE      4095   // range of position cmd DAC shifted 2 bits

// intial values
#define ZCTRL_INIT  0x20;
            //!SYNC_MODE,!TESTn,!PKDET_MODE,!RST_FOCUSn,RST_INTEGN
#define INIT_THRESH  0x16;
```

```
//*******************************************************************
//*********************** PRAGMAS ***********************************
//*******************************************************************
#pragma net_buf_in_count  7
#pragma net_buf_out_count 7


//*******************************************************************
//*********************** I/O DECLARATIONS **************************
//*******************************************************************
IO_0  output   byte   WRITE_DATA;    // parallel port
IO_0  input    byte   READ_DATA;

IO_8   output  bit    RD_WRn;        // ReaD/not-WRite
IO_9   output  bit    ADDR_DATAn;    // ADDRess/not-DATA
IO_10  output  bit    STBn=1;        // low-true STroBe


//*******************************************************************
//*********************** NETWORK VARIABLES *************************
//*******************************************************************
network output polled unsigned SW_Rev = SWR;   // software version
network output unsigned ResetSts;      // for indicating reset
network input  unsigned OpMode=NULL_MODE;      // for remote proc. calls
network output unsigned OpStatus;      // response to calls
network input  unsigned OpParArray[2];  // Operation parameters
network output unsigned OpStsArray[2];  // Extra Status info

network input   unsigned AFocusThresh;  // 0-255 for Autofocus threshold
network input   unsigned long ZPosCmd;   // Position Command 0-4095
network input   unsigned Z_Control;     // Control byte
network input   unsigned MuxSel;       // ADC mux selector
network input   unsigned ZStep;        // step size for vol acquire
network input   unsigned NumFrames;  // # frames for volume
network output polled unsigned Z_Status;          // Status byte
network output polled unsigned long PosStatus;   // more status

struct PMTARRAY               // define structure to hold pmt values   tvt
{
   unsigned    pmt[16];
};

// declare 16 network variables as structure of PMTARRAY type
// to hold 256 pmt readings (one every 16 steps of the fast z stage tvt

network output polled struct PMTARRAY pmtvals1;
network output polled struct PMTARRAY pmtvals2;
network output polled struct PMTARRAY pmtvals3;
network output polled struct PMTARRAY pmtvals4;
network output polled struct PMTARRAY pmtvals5;
network output polled struct PMTARRAY pmtvals6;
network output polled struct PMTARRAY pmtvals7;
network output polled struct PMTARRAY pmtvals8;
network output polled struct PMTARRAY pmtvals9;
network output polled struct PMTARRAY pmtvals10;
network output polled struct PMTARRAY pmtvals11;
network output polled struct PMTARRAY pmtvals12;
far network output polled struct PMTARRAY pmtvals13;
far network output polled struct PMTARRAY pmtvals14;
far network output polled struct PMTARRAY pmtvals15;
far network output polled struct PMTARRAY pmtvals16;

//*****************************************************************
//********************** GLOBAL VARIABLES **************************
//*****************************************************************
```

```
//*******************************************************************
//********************** FUNCTION DEFINITIONS **********************
//*******************************************************************
unsigned short ReadPort (unsigned short TargetAddr)
// read a port, including set up address
  {
  unsigned short result;

  io_set_direction (WRITE_DATA, IO_DIR_OUT);
  io_out (ADDR_DATAn, ADDR);            // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, TargetAddr | MuxSel);     // write address
  io_out (ADDR_DATAn, DATA);            // set ADDR_DATAn to DATA
  io_set_direction (READ_DATA, IO_DIR_IN);
  io_out (RD_WRn, READ);              // set RD_WRn for READ
  io_out (STBn, 0);              // assert STBn
  result = io_in (READ_DATA);         // read data
  io_out (STBn, 1);              // de-assert STBn
  return (result);              // return result
  }


void WritePort (unsigned short TargetAddr, unsigned short WriteData)
// write to a port, including set up address
  {

  io_set_direction (WRITE_DATA, IO_DIR_OUT);
  io_out (ADDR_DATAn, ADDR);               // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, TargetAddr | MuxSel);   // write address
  io_out (ADDR_DATAn, DATA);               // set ADDR_DATAn to DATA
  // no need to change port direction for write
  io_out (RD_WRn, WRITE);               // set RD_WRn for WRITE
  io_out (WRITE_DATA, WriteData);          // set up data
  io_out (STBn, 0);                  // assert STBn
  io_out (STBn, 1);                  // de-assert STBn
```

```
    }


unsigned FastRead (void)
// read a port, without address set up
  {
  unsigned result;

  io_out (STBn, 0);                  // assert STBn
  result = io_in (READ_DATA);             // read data
  io_out (STBn, 1);                  // de-assert STBn
  return (result);                 // return result
  }


void FastWrite (unsigned short WriteData)
// write to a port, without address set up
  {

  io_out (WRITE_DATA, WriteData);       // set up data
  io_out (STBn, 0);                  // assert STBn
  io_out (STBn, 1);                  // de-assert STBn
  }


void SetAddressRead (unsigned short TargetAddr)
// Setup address for fast read
  {

  io_set_direction (WRITE_DATA, IO_DIR_OUT);
  io_out (ADDR_DATAn, ADDR);            // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, TargetAddr | MuxSel);      // write address
  io_out (ADDR_DATAn, DATA);            // set ADDR_DATAn to DATA
  io_set_direction (READ_DATA, IO_DIR_IN);
  io_out (RD_WRn, READ);              // set RD_WRn for READ
  }


void SetAddressWrite (unsigned short TargetAddr)
// Setup address for fast read
  {

  io_set_direction (WRITE_DATA, IO_DIR_OUT);
  io_out (ADDR_DATAn, ADDR);          // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, TargetAddr | MuxSel);     // write address
  io_out (ADDR_DATAn, DATA);          // set ADDR_DATAn to DATA
  // no need to change port direction for write
  io_out (RD_WRn, WRITE);             // set RD_WRn for WRITE
  }


void SetCtrl (unsigned NewCtrl)
// sets the control word
  {
  Z_Control = NewCtrl;
  WritePort (CTRL_REG, NewCtrl);
  }


void MoveZ (unsigned long target)
// move the Z-axis to the current commanded position (ZPosCmd)
  {
  if (target > DAC_RANGE)
    target = DAC_RANGE;
  ZPosCmd = (target< <DAC_SHIFT);
  WritePort (POS_CMD_LO, (unsigned)(ZPosCmd & 0xFF)); // set position
  WritePort (POS_CMD_HI, (unsigned)((ZPosCmd > > 8) & 0xFF));
  ZPosCmd = target;
  }

void ResetZSync(void)
{
  io_set_direction (WRITE_DATA, IO_DIR_OUT);      // set port to out
  io_out (ADDR_DATAn, ADDR);                      // set ADDR_DATAn to ADDR
  io_out (WRITE_DATA, (~RST_ZSYNCn) | MuxSel);    // write address
  io_out(RD_WRn, READ);                           // set read line
  io_out (STBn, 0);                     // assert STBn
  io_out (STBn, 1);                     // de-assert STBn
}

unsigned long AFFast(void)    // tvt
//  Fast Autofocus Function: This funciton does a coarse then a fine autofocus
{
  unsigned short NextVal, BestVal;  // ADC values
  unsigned long BestPos;            // position of peak signal
  unsigned long AvgVal;             // average value storage
  unsigned long FineCenter;         // center of fine focus range
  unsigned long Move_Delta;         // steps per moove
  unsigned long step;               // z stage step loop counter
  unsigned m;                       // array member
  unsigned l;                       // loop counter
  static int FocOff;                // Focus offset
  unsigned int StepSize;            // step size during coarse focus

  // Set FocOff and StepSize
     FocOff= OpParArray[0];
     StepSize = 128;

  SetCtrl (Z_Control & (~SYNC_MODE));  // async mode
  // move to the extreme lower limit
  MoveZ (0);                        // Start at the bottom
  delay(1977);                      // wait 50 ms
```

RECTIFIED SHEET (RULE 91)

```
// ***** coarse autofocus *****
// step through entire Z range with large steps
// checking for peak PMT and storing values
   AvgVal=0;
   MuxSel = SEL_VIDEO;
   AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
   AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
   AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
   AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
   NextVal=(short)(AvgVal/4);
   BestVal=NextVal;                  // Clear Largest value
   BestPos=0;                      // Clear Best Position

   for (step=0; step <= 4096; step +=StepSize)  // Entire range in steps of 128
   {
      if(step == 0) step = 1;
      MoveZ(step-1);                 // Move 0 to 4095
      watchdog_update();
      AvgVal=0;
      MuxSel = SEL_VIDEO;
      AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
      AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
      AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
      AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
      NextVal=(short)(AvgVal/4);

      if (NextVal > BestVal)          // If Current value is larger
      {                             // than the best (largest) value
         BestPos=step;              // this is the best position
         BestVal=NextVal;            // Update best value
      }
   }

// Move to start value for fine autofocus and wait
```

```
    if (BestPos > 129)
       MoveZ (BestPos-128-1); // Start at the bottom
    else
       MoveZ (1);
    delay(395);                    // wait 10 ms

// Set up comparison values
    AvgVal=0;
    MuxSel = SEL_VIDEO;
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
    NextVal=(short)(AvgVal/4);
    BestVal=NextVal;                    // Clear Largest value

// ***** fine autofocus *****
// using position from coarse focus
    FineCenter=BestPos; // Set center of focus to the best coarse focus pos.
    if(FineCenter<129)  // Make sure it's valid (at least 129
       FineCenter=129;
    for (step = FineCenter-128; step <= FineCenter+128; step +=8)
    {
       MoveZ(step-1);                 // Move through fine af range
       watchdog_update();
       AvgVal=0;
       MuxSel = SEL_VIDEO;
       AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
       AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
       AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
       AvgVal=AvgVal+ReadPort(ADC_ADDR); // Store current PMT value
       NextVal=(short)(AvgVal/4);

       if (NextVal > BestVal)          // If Current value is larger
```

```
        {                               // than the best (largest) value
          BestPos=step;                   // this is the best position
          BestVal=NextVal;                  // Update best value
        }
      }

      OpStatus = BestVal;                   // send the largest value out
      MoveZ(BestPos-1-FocOff);                // move to the best position
      return BestPos-1-FocOff;              // return the best positon
    }

    unsigned long AFPeakSync(void)    // tvt
    // Fast Autofocus Function with peak detection and synchronous operation
    // Note that in synchronous mode, when a write to the position command
    // registers is executed, the stage does not receive the command until the
    // next ZSYNC, and the stage does not settle until a few msecs beyond that.
    // Do not use the 'MoveZ' routine while in synchronous mode because it
    // reverts to asynchronous mode.
    // The coarse pass of this Af routine operates in asynchronous mode,
    // stepping as quickly as possible, and not paying attention to where
    // the scanners are during sampling. In the fine pass, however, the page
    // scanner is run at a high
    // rate (>100Hz) and allow one page scanner cycle for the FastZ to settle
    // before doing data acquisition on the next. The routine therefore
    // spends two page scanner cycles at each Z position. Furthermore, in
    // peak detector mode, an ADC conversion is automatically started when
    // ZSYNC occurs, and the peak detector is automatically reset. So when
    // the routine waits for ZSYNC and then reads the ADC, the ADC reading
    // applies to the peak detected signal from the previous page scan.
    // Also note that resetting ZSYNC also resets the ADC, so the ADC should
    // be read before resetting ZSYNC. In peak detector mode, the ADC is
    // controlled ONLY by ZSYNC; there is no way to do an asynchronous
    // conversion without resetting the peak detector mode bit. The peak
    // detector is used to obtain a representation of the highest intensity
```

```
// that occurred in a video frame at a given Z elevation. The intention
// is to repeatibly focus on the layer with the highest reflectivity, and
// use programmable offsets (FocOff) so that the user can select any layer
// to focus on.
// One final note: this routine will time out if the page scanner
// rate is not fast enough, because 'watchdog_update' is used sparingly.
{
    unsigned short NextVal, BestVal;// ADC values
    unsigned long BestPos;          // position of peak signal
    unsigned long step;             // z stage step loop counter
    unsigned long RunningStart, AcquirePos,ZPosShift;
    boolean MoveDown;
    unsigned long RUN_START;
    unsigned m;                     // array member
    unsigned l;                     // loop counter
    static int FocOff;              // Focus offset

    FocOff = OpParArray[0];
    RUN_START = 0;
    // move to the extreme lower limit
    MoveZ (0);                      // Start at the bottom
    delay (1977);                   // 50ms delay

    // ***** coarse autofocus *****
    // step through entire Z range with large steps
    // checking for peak PMT and storing values
    MuxSel = SEL_VIDEO;
    NextVal = 0;
    BestVal=0;                      // Clear Largest value
    BestPos=0;                      // Clear Best Position

    // assumes have moved to start position, use async mode for coarse pass
    SetCtrl (Z_Control & (~SYNC_MODE) & (~PKDET_MODE));
```

```
for (step = 0; step < = DAC_RANGE + 1; step + = COARSE_STEP)
{
   if (step > = DAC_RANGE + 1) step = DAC_RANGE;
   MoveZ (step);
   NextVal = ReadPort (ADC_ADDR);
   if (NextVal > BestVal)          // If Current value is larger
   {                             // than the best (largest) value
      BestPos=step;              // this is the best position
      BestVal=NextVal;           // Update best value
   }
}

OpStsArray[0] = (short)(BestPos & 0xFF); // return value for diagnostics
OpStsArray[1] = (short)((BestPos & 0xFF00) > > 8);
// Move to start value for fine autofocus
// top two possible positions from coarse pass are special
// and so is bottom position
if (BestPos > = DAC_RANGE + 1 - FINE_RANGE) { // top 2 positions
  RunningStart = DAC_RANGE + 1 - (2 * FINE_RANGE) - RUN_START;
  AcquirePos = DAC_RANGE + 1 - (2 * FINE_RANGE);
  MoveDown = FALSE; }
else if (BestPos < FINE_RANGE) {            // bottom position
  RunningStart = (2 * FINE_RANGE) + RUN_START;
  AcquirePos = (2 * FINE_RANGE);
  MoveDown = TRUE; }                        // other
else {
  RunningStart = BestPos + FINE_RANGE + RUN_START;
  AcquirePos = BestPos + FINE_RANGE;
  MoveDown = TRUE; }
MoveZ (RunningStart);

// fine pass in peak detect/sync mode

MuxSel = SEL_PKDET;                // move to sync, pkdet modes
```

```
SetCtrl (Z_Control | SYNC_MODE | PKDET_MODE );
ResetZSync();
while ((ReadPort(STS_REG) & ZSYNCDn) != 0)  // wait for sync
  ;
ResetZSync();
// Set up comparison values
NextVal=0;
BestVal=0;                          // clear Largest value

// ***** fine autofocus *****
// using position from coarse focus
if (MoveDown == TRUE)                   // downward pass
  {
  for (step = RunningStart;          // running start
          step > AcquirePos;
          step -= FINE_STEP)
    {
    // wait for ZSYNC (dummy sync while stage in transit)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
      ;
    // note that the following move happens on the next ZSYNC
    ZPosShift = step<<DAC_SHIFT;
    WritePort (POS_CMD_LO, (unsigned int)(ZPosShift  & 0xFF));
    WritePort (POS_CMD_HI, (unsigned int)(ZPosShift >> 8) & 0xFF);
    // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
    ResetZSync();
    // wait for sync (peak detector is acquiring)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
      ;
    // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
    ResetZSync();
    }
  for (step = AcquirePos;           // fine pass data acq
        (step >= AcquirePos - (2 * FINE_RANGE)) &&
            (step < DAC_RANGE);        // because unsigned
            step -= FINE_STEP)
        {
        // wait for ZSYNC (dummy sync while stage in transit)
        while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
          ;
        // note that the following move happens on the next ZSYNC
        ZPosShift = step<<DAC_SHIFT;
        WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
        WritePort (POS_CMD_HI, (unsigned int)(ZPosShift>> 8) & 0xFF);
        // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
        ResetZSync();
        // wait for sync (peak detector is acquiring)
        while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
          ;
        NextVal=ReadPort(ADC_ADDR);
        if (NextVal > BestVal)            // If Current value is larger
          {                               // than the best (largest) value
          BestPos = step  + FINE_STEP;    // this is the best position
          BestVal = NextVal;              // Update best value
          }
        // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
        ResetZSync();
        }
    // now process the last step (because of pipeline)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)  // wait for last step
      ;                                         // to settle
     ResetZSync();
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)  // wait for peak det
      ;                                         // to load up
    NextVal=ReadPort(ADC_ADDR);
    if (NextVal > BestVal)            // If Current value is larger
      {                               // than the best (largest) value
      BestPos=step + FINE_STEP;       // this is the best position
    if (BestPos >= DAC_RANGE + 1) BestPos = 0;
      BestVal=NextVal;            // Update best value
    }
  }
else                                  // upward pass
  {
  for (step = RunningStart;           // running start
        step < AcquirePos;
        step += FINE_STEP)
    {
    // wait for ZSYNC (dummy sync while stage in transit)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
      ;
    // note that the following move happens on the next ZSYNC
    ZPosShift = step<<DAC_SHIFT;
    WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
    WritePort (POS_CMD_HI, (unsigned int)(ZPosShift>> 8) & 0xFF);
    // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
    ResetZSync();
    // wait for sync (peak detector is acquiring)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
      ;
    // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
  ResetZSync();
    }
 for (step = AcquirePos;             // fine pass data acq
        step <= AcquirePos + (2 * FINE_RANGE);
        step += FINE_STEP)
    {
    if (step >= (DAC_RANGE + 1)) step = DAC_RANGE;
    // wait for ZSYNC (dummy sync while stage in transit)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
      ;
    // note that the following move happens on the next ZSYNC
```

```
        ZPosShift = step<<DAC_SHIFT;
        WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
        WritePort (POS_CMD_HI, (unsigned int)(ZPosShift>> 8) & 0xFF);
        // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
        ResetZSync();
        // wait for sync (peak detector is acquiring)
        while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
          ;
        NextVal=ReadPort(ADC_ADDR);
        if (NextVal > BestVal)          // If Current value is larger
          {                             // than the best (largest) value
          BestPos = step - FINE_STEP;   // this is the best position
          BestVal = NextVal;            // Update best value
          }
        // reset the ZSYNC flip flop for next time (also resets ADC CNVRT)
         ResetZSync();
        }
    // now process the last step (because of pipeline)
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)  // wait for last step
      ;                                         // to settle
    ResetZSync();
    while ((ReadPort(STS_REG) & ZSYNCDn) != 0)  // wait for peak det
      ;                                         // to load up
    NextVal=ReadPort(ADC_ADDR);
    if (NextVal > BestVal)          // If Current value is larger
      {                             // than the best (largest) value
      BestPos = step - FINE_STEP;   // this is the best position
      if (BestPos >= DAC_RANGE+1) BestPos = DAC_RANGE;
        BestVal=NextVal;            // Update best value
      }
    }

ResetZSync();
SetCtrl (Z_Control & (~PKDET_MODE));
```

```
    OpStatus = BestVal;          // send the largest value out
    ZPosShift = (BestPos - FocOff) < < DAC_SHIFT;   // move to the best position
    WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
    WritePort (POS_CMD_HI, (unsigned int)(ZPosShift> > 8) & 0xFF);
    return (BestPos - FocOff);     // return the best positon
  }

  unsigned long AFPMTVals(void)    // tvt
  // Function to step through entire fast z stage range at 16 counts per step
  // storing PMT values at each step in network variable structures (16 structures
  // of 16 element arrays). Stage is returned to fast z position which gives the
  // highest PMT value.
  {
    unsigned short NextVal, BestVal;  // ADC values
    unsigned long BestPos;           // position of peak signal
    unsigned long Move_Delta;          // steps per moove
    unsigned long step;             // z stage step loop counter
    unsigned m;                     // array member
    static int FocOff;              // Focus offset

    SetCtrl (Z_Control & (~SYNC_MODE));   // async mode
    // move to the extreme lower limit
    MoveZ (0);
    delay(1977);                     // wait 50 ms

  // step through entire Z range checking for peak PMT and storing values

    NextVal=ReadPort(ADC_ADDR);
    BestVal=NextVal;
    BestPos=0;

    for (step=0; step < = 4096; step + =16)
    {
      if(step = = 0) step = 1;
```

```
MoveZ(step-1);
watchdog_update();
NextVal = ReadPort(ADC_ADDR);

if (NextVal > BestVal)
{
   BestPos=step;
   BestVal=NextVal;
}

/* Write PMT Values to nv's */

if (step < 256)  // use structure 1
{
   m = (short)(step/16);
   pmtvals1.pmt[m] = NextVal;
}

else if ((step >= 256) && (step < 512)) // use structure 2
{
   m = (short)((step-256)/16);
   pmtvals2.pmt[m]=NextVal;
}

else if ((step >= 512) && (step < 768)) // use structure 3
{
   m= (short)((step-512)/16);
   pmtvals3.pmt[m]=NextVal;
}

else if ((step >= 768) && (step < 1024)) // use structure 4
{
   m =  (short)((step - 768) / 16);
   pmtvals4.pmt[m]=NextVal;
```

```
}

else if ((step >= 1024) && (step < 1280)) // use structure 5
{
   m =  (short)((step - 1024) / 16);
   pmtvals5.pmt[m]=NextVal;
}

else if ((step >= 1280) && (step < 1536)) // use structure 6
{
   m =  (short)((step - 1280) / 16);
   pmtvals6.pmt[m]=NextVal;
}

else if ((step >= 1536) && (step < 1792)) // use structure 7
{
   m = (short)((step - 1536) / 16);
   pmtvals7.pmt[m]=NextVal;
}

else if ((step >= 1793) && (step < 2048)) // use structure 8
{
   m = (short)((step - 1793) / 16);
   pmtvals8.pmt[m]=NextVal;
}

else if ((step >= 2048) && (step < 2304)) // use structure 9
{
   m = (short)((step - 2048) / 16);
   pmtvals9.pmt[m]=NextVal;
}

else if ((step >= 2304) && (step < 2560)) // use structure 10
{
    m = (short)((step - 2304) / 16);
    pmtvals10.pmt[m]=NextVal;
}

else if ((step >= 2560) && (step < 2816)) // use structure 11
{
    m = (short)((step - 2560) / 16);
    pmtvals11.pmt[m]=NextVal;
}

else if ((step >= 2816) && (step < 3072)) // use structure 12
{
    m = (short)((step - 2816) / 16);
    pmtvals12.pmt[m]=NextVal;
}

else if ((step >= 3072) && (step < 3328)) // use structure 13
{
    m = (short)((step - 3072) / 16);
    pmtvals13.pmt[m]=NextVal;
}

else if ((step >= 3328) && (step < 3584)) // use structure 14
{
    m = (short)((step - 3328) / 16);
    pmtvals14.pmt[m]=NextVal;
}

else if ((step >= 3584) && (step < 3840)) // use structure 15
{
    m = (short)((step - 3584) / 16);
    pmtvals15.pmt[m]=NextVal;
}
```

```
        else // if ((step >= 3840) && (step < 4096)) // use structure 16
        {
           m = (short)((step - 3840) / 16);
           pmtvals16.pmt[m]=NextVal;
        }


    }

    OpStatus = BestVal;             // send the largest value out
    MoveZ(BestPos-1-FocOff);          // move to the best position
    return BestPos-1-FocOff;        // return the best positon
}


unsigned ReadVidLoop(void)
{
   // This functions reads the pmt video value 'a' times to find an average at
   // points in the Fine Z travel with 'l' step size. a and l values are
   // updated by network variables to give the desired time for the loop
   // which is about 1 second in the LIS 1000

   unsigned   l,a;          // loop and average counter
   unsigned   MaxVid;          // Maximum Video Value
   unsigned   CurVid;         // Current Video Value
   unsigned long AvgVid;       // Average Video
   int        MaxStatus;     // Satus of MaxVal
   unsigned   Avg;            // Averaging value
   unsigned   Count;          // loop count

// Initialize
   MuxSel=SEL_VIDEO;            // Select Video;
   SetCtrl(Z_Control & (~SYNC_MODE) & (~PKDET_MODE));
   CurVid=ReadPort(ADC_ADDR);  // Get Current Video Value
```

```
    MaxVid=CurVid;            // Set Maximum to starting value
    OpStatus=0;               // Clear OpStatus
    OpStsArray[0]=0;          // Clear OpParArray[0]
    OpStsArray[1]=0;          // Clear OpParArray[1]
    Count=OpParArray[0];      // Set loop count
    Avg=OpParArray[1];        // Set averaging

    if (Avg == 0)
      Avg=1;                  // Make sure averaging is in range

    if (Count < 64)
      Count=64;               // Make sure count is in range



// PMT Read Loop
    for (l=0;l<=Count;l++)
    // for 64 counts
    {
      AvgVid=0;

      for (a=0; a<Avg; a++)
      AvgVid = AvgVid + ReadPort(ADC_ADDR); // Get Current Video Level
      CurVid = (short)(AvgVid/a);       // averaged over a

      if (CurVid > MaxVid)        // If Current level is greater
      {
        MaxVid=CurVid;            // than max, reset max.
        MaxStatus=1;              // status of maximum.
      }

      // Store Video Level

      if (l<ARRAYSIZE)
```

```
{
   pmtvals1.pmt[l%ARRAYSIZE]=CurVid;       // Store Current Video
   if (MaxStatus==1)                       // if this is max
   {
      OpStsArray[0]=0;                     // Set Par0 to array
      OpStsArray[1]=1%ARRAYSIZE;           // Set Par1 to element
   }
}
else if ((l>=ARRAYSIZE) && (l<ARRAYSIZE*2))
{
   pmtvals2.pmt[l%ARRAYSIZE]=CurVid;       // Store Current Video
   if (MaxStatus==1)                       // if this is max
   {
      OpStsArray[0]=1;                     // Set Par0 to array
      OpStsArray[1]=1%ARRAYSIZE;           // Set Par1 to element
   }
}

else if ((l>=ARRAYSIZE*2) && (l<ARRAYSIZE*3))
{
   pmtvals3.pmt[l%ARRAYSIZE]=CurVid;       // Store Current Video
   if (MaxStatus==1)                       // if this is max
   {
      OpStsArray[0]=2;                     // Set Par0 to array
      OpStsArray[1]=1%ARRAYSIZE;           // Set Par1 to element
   }
}
else if ((l>=ARRAYSIZE*3) && (l<ARRAYSIZE*4))
{
   pmtvals4.pmt[l%ARRAYSIZE]=CurVid;       // Store Current Video
   if (MaxStatus==1)                       // if this is max
   {
      OpStsArray[0]=3;                     // Set Par0 to array
      OpStsArray[1]=1%ARRAYSIZE;           // Set Par1 to element
```

```
        }
    }
    else if ((l> =ARRAYSIZE*4) && (l<ARRAYSIZE*5))
    {
        pmtvals5.pmt[l%ARRAYSIZE]=CurVid;     // Store Current Video
        if (MaxStatus==1)                     // if this is max
        {
            OpStsArray[0]=4;                  // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> =ARRAYSIZE*5) && (l<ARRAYSIZE*6))
    {
        pmtvals6.pmt[l%ARRAYSIZE]=CurVid;     // Store Current Video
        if (MaxStatus==1)                     // if this is max
        {
            OpStsArray[0]=5;                  // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> =ARRAYSIZE*6) && (l<ARRAYSIZE*7))
    {
        pmtvals7.pmt[l%ARRAYSIZE]=CurVid;     // Store Current Video
        if (MaxStatus==1)                     // if this is max
        {
            OpStsArray[0]=6;                  // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l> =ARRAYSIZE*7) && (l<ARRAYSIZE*8))
    {
        pmtvals8.pmt[l%ARRAYSIZE]=CurVid;     // Store Current Video
        if (MaxStatus==1)                     // if this is max
        {
            OpStsArray[0]=7;                    // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l>=ARRAYSIZE*8) && (l<ARRAYSIZE*9))
    {
        pmtvals9.pmt[l%ARRAYSIZE]=CurVid;       // Store Current Video
        if (MaxStatus==1)                       // if this is max
        {
            OpStsArray[0]=8;                    // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l>=ARRAYSIZE*9) && (l<ARRAYSIZE*10))
    {
        pmtvals10.pmt[l%ARRAYSIZE]=CurVid;      // Store Current Video
        if (MaxStatus==1)                       // if this is max
        {
            OpStsArray[0]=9;                    // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l>=ARRAYSIZE*10) && (l<ARRAYSIZE*11))
    {
        pmtvals11.pmt[l%ARRAYSIZE]=CurVid;      // Store Current Video
        if (MaxStatus==1)                       // if this is max
        {
            OpStsArray[0]=10;                   // Set Par0 to array
            OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
        }
    }
    else if ((l>=ARRAYSIZE*11) && (l<ARRAYSIZE*12))
    {
        pmtvals12.pmt[l%ARRAYSIZE]=CurVid;      // Store Current Video
        if (MaxStatus==1)                   // if this is max
        {
           OpStsArray[0]=11;                 // Set Par0 to array
           OpStsArray[1]=l%ARRAYSIZE;         // Set Par1 to element
        }
     }
     else if ((l>=ARRAYSIZE*12) && (l<ARRAYSIZE*13))
     {
        pmtvals13.pmt[l%ARRAYSIZE]=CurVid;    // Store Current Video
        if (MaxStatus==1)                   // if this is max
        {
           OpStsArray[0]=12;                 // Set Par0 to array
           OpStsArray[1]=l%ARRAYSIZE;         // Set Par1 to element
        }
     }
     else if ((l>=ARRAYSIZE*13) && (l<ARRAYSIZE*14))
     {
        pmtvals14.pmt[l%ARRAYSIZE]=CurVid;    // Store Current Video
        if (MaxStatus==1)                   // if this is max
        {
           OpStsArray[0]=13;                 // Set Par0 to array
           OpStsArray[1]=l%ARRAYSIZE;         // Set Par1 to element
        }
     }
     else if ((l>=ARRAYSIZE*14) && (l<ARRAYSIZE*15))
     {
        pmtvals15.pmt[l%ARRAYSIZE]=CurVid;    // Store Current Video
        if (MaxStatus==1)                   // if this is max
        {
           OpStsArray[0]=14;                 // Set Par0 to array
           OpStsArray[1]=l%ARRAYSIZE;         // Set Par1 to element
        }
     }
     else // if ((l>=ARRAYSIZE*15) && (l<ARRAYSIZE*16))
```

```
            {
               pmtvals16.pmt[l%ARRAYSIZE]=CurVid;     // Store Current Video
               if (MaxStatus==1)                 // if this is max
               {
                  OpStsArray[0]=15;               // Set Par0 to array
                  OpStsArray[1]=l%ARRAYSIZE;          // Set Par1 to element
               }
            }

            MaxStatus=0;
            watchdog_update();
         }

      return MaxVid;

      }

      unsigned CheckParms(void)
        {
        signed long FinalPos;
        unsigned short RetVal;

        FinalPos = ZPosCmd - (NumFrames-1)*ZStep;
        if (FinalPos < 0)
          RetVal = UNDERFLOW;
        else
          RetVal = RDY_FOR_VOL;
        return (RetVal);
        }

      unsigned SelfTest (void)
        {
        unsigned AllErrors, OldCtrl;
```

```
AllErrors = 0;
OldCtrl = Z_Control;
// turn all test bits off
SetCtrl ( Z_Control & (~(SPARE_TEST1 | SPARE_TEST2 | TEST_LEDn |
                                                    PKDET_MODE)));
// test
if ((ReadPort (STS_REG) & (TEST_IN1 | TEST_IN2 | PKDET_MODE |
                                                    LED_TST_IN)) != 0)
  AllErrors = AllErrors | 0x1;
// turn spare test 1 on
SetCtrl ( Z_Control | SPARE_TEST1);
// test
if ((ReadPort (STS_REG) & TEST_IN1) != TEST_IN1)
  AllErrors = AllErrors | 0x2;
// turn spare test 1 off, spare test 2 on
SetCtrl ( (Z_Control | SPARE_TEST2) & (~SPARE_TEST1));
// test
if ((ReadPort (STS_REG) & (TEST_IN1 | (TEST_IN2)) !=TEST_IN2)
  AllErrors = AllErrors | 0x4;
// turn spare test 2 off, peak det on
SetCtrl ( (Z_Control | PKDET_MODE) & (~SPARE_TEST2));
// test
if ((ReadPort (STS_REG) & (TEST_IN1 | TEST_IN2 | PKDET_IN))
                                                    !=PKDET_IN)
  AllErrors = AllErrors | 0x8;
// turn peak det off, led on
SetCtrl ( (Z_Control | TEST_LEDn) & (~PKDET_MODE)):
// test
if ((ReadPort (STS_REG) & (TEST_IN1 | TEST_IN2 | PKDET_IN |
                                   LED_TST_IN)) !=LED_TST_IN)
  AllErrors = AllErrors | 0x10;
// set control to original value
SetCtrl (OldCtrl);
return (AllErrors);
```

```
  }

//*****************************************************************
//************************ WHEN STATEMENTS ***********************
//*****************************************************************
when (reset)
  {
  unsigned long i,Zmove,Thresh;
  short dummy;

  ResetSts = 0;                     // indicate reset to host
  SetCtrl(ZCTRL_INIT);
  AFocusThresh = INIT_THRESH;
  MoveZ (0);
  MuxSel = SEL_VIDEO;

  if (((Z_Status = ReadPort (STS_REG)) & TEST_JMPRn) == 0) // if test mode
    {
    // exercise everything (SYNC_MODE,RST_FOCUS*,RST_INTEG*,
                                                          PKDET_MODE
    // W_CNTL*, W_THRESH*, W_DACLO*, W_DACHI*, R_STS*, R_ADC*,
                                                          RST_ZSYNC*)

    SetCtrl (0);
    while (TRUE)                    // until next reset
       {
         watchdog_update();
         SetCtrl(~Z_Control);       // toggle control bits
            dummy=ReadPort (STS_REG);
            Z_Status = ReadPort(ADC_ADDR);
         for(i=0; i<DAC_RANGE; i+=32)
         {
            //AFocusThresh++;
            //ZPosCmd++;
```

```
                Zmove=i< <DAC_SHIFT;
                WritePort(POS_CMD_LO, ((unsigned int)(Zmove & 0xFF)));   //
                WritePort(POS_CMD_HI, ((unsigned int) ((Zmove > > 8) & 0xFF)));
                                                                        //& 0xFF
                WritePort(THRESH_DAC, (unsigned int)(i> >4));   // set threshold
                post_events();
             }
             for(i=DAC_RANGE; i> =65; i-=32)
             {
                //AFocusThresh+ +;
                //ZPosCmd+ +;
                Zmove=i< <DAC_SHIFT;                //
                WritePort(POS_CMD_LO, ((unsigned int)(Zmove)& 0xFF));
                WritePort(POS_CMD_HI, ((unsigned int) ((Zmove > > 8) & 0xFF)));  //
                WritePort(THRESH_DAC, (unsigned int)(i> >4));   // set threshold
                post_events();
             }
          }
       }
    else
       {
       SetCtrl (TEST_LEDn); // set control bits: async mode,
                            // reset all FF's, reset integ,
                            // turn off test LED
       WritePort (THRESH_DAC, AFocusThresh);
       SetCtrl (TEST_LEDn | RST_FOCUSn | RST_INTEGn);
       }

    }

 when (nv_update_occurs (AFocusThresh))
 // Update the auto focus threshold DAC
    {
    WritePort (THRESH_DAC, AFocusThresh);
    }


 when (nv_update_occurs (ZPosCmd))
 // update the position command DAC
    {
    if (ZPosCmd > DAC_RANGE)
       ZPosCmd = DAC_RANGE;
    SetCtrl(Z_Control & (~SYNC_MODE)); //async mode
    MoveZ (ZPosCmd);
    }


 when (nv_update_occurs (Z_Control))
 // update the control word
    {
    WritePort (CTRL_REG, Z_Control);
    }


 // ****************************************************************
 // *************** remote procedure calls ************************
 // ****************************************************************
 // when (nv_update_occurs) is not used here so that OpMode can
 // be monitored within the when clauses, specifically to break
 // out of the MONITOR_FOCUS mode.  OpMode is changed to NULL_MODE
 // at the end of each clause to ensure that they are only invoked
 // once.

 when (OpMode == SELF_TEST)
    {
    OpStatus = SelfTest();
    OpMode = NULL_MODE;
    }
```

```
when (OpMode == AF_FAST)
   {
   // Perform fast autofocus function
   MuxSel = SEL_VIDEO;
   PosStatus = AFFast();
   OpMode = NULL_MODE;
   }

when (OpMode == AF_PEAK_SYNC)
   {
   // Perform fast autofocus function
   MuxSel = SEL_VIDEO;
   PosStatus = AFPeakSync();
   OpMode = NULL_MODE;
   }

when (OpMode == AF_PMTVALS)
   {
   // Perform autofocus function which writes PMT values
   MuxSel = SEL_VIDEO;
   PosStatus = AFPMTVals();
   OpMode = NULL_MODE;
   }

when (OpMode == READ_VIDEO)
  // Read ADC on stated mux channel)
  {
  MuxSel = SEL_VIDEO;
  OpStatus = ReadPort (ADC_ADDR);
  OpMode = NULL_MODE;
  }

when (OpMode == RD_VID_LP)
   // call function to read PMT values
```

```
        {
        MuxSel = SEL_VIDEO;
        OpStatus = ReadVidLoop();
        OpMode = NULL_MODE;
        }
      when (OpMode == READ_HV)
       // Read ADC on stated mux channel)
       {
       MuxSel = SEL_HV;
       OpStatus = ReadPort (ADC_ADDR);
       OpMode = NULL_MODE;
          }

      when (OpMode == READ_FDBK)
       // Read ADC on stated mux channel)
       {
       MuxSel = SEL_FDBK;
       OpStatus = ReadPort (ADC_ADDR);
       OpMode = NULL_MODE;
       }

     when (OpMode == SET_FOR_VOL)
       // setup for volume acquisition)
       {
       unsigned FrameCtr;
       unsigned long ZPosShift;

       if ((OpStatus = CheckParms()) == RDY_FOR_VOL)
         {
         // clear ZSYNC FF, wait for syncs and step when they come
         // assumes have moved to start position (closest to objective)
         // assumes ZSYNCs have been stopped, so clear ZSYNC FF and wait

         //SetCtrl (Z_Control & (~RST_ZSYNCn));          // clear ZSYNC
        ResetZSync();
        SetCtrl (Z_Control | SYNC_MODE);  // go to sync mode
        // load DAC register with current position so it stays put
        // on first ZSYNC
        ZPosShift = ZPosCmd << DAC_SHIFT;
        WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
        WritePort (POS_CMD_HI, (unsigned int)(ZPosShift >> 8) & 0xFF);
        // alert host that ready for volume
        OpStatus = RDY_FOR_VOL;
        post_events();
        // take the volume)
        for (FrameCtr = 0; FrameCtr < NumFrames; FrameCtr++)
          {
          // wait for ZSYNC
          while ((ReadPort(STS_REG) & ZSYNCDn) != 0)
            ;
          // reset the ZSYNC flip flop
          ResetZSync();
          // step down
          ZPosCmd = (ZPosCmd - ZStep);
          ZPosShift = ZPosCmd << DAC_SHIFT;
          WritePort (POS_CMD_LO, (unsigned int)(ZPosShift & 0xFF));
          WritePort (POS_CMD_HI, (unsigned int)(ZPosShift >> 8) & 0xFF);
          watchdog_update();              // prevent reset
          }
      SetCtrl (Z_Control & (~SYNC_MODE)); // async mode
      OpStatus = VOLUME_COMPL;
      }
    PosStatus = ZPosCmd;
    OpMode = NULL_MODE;
    }

  when (OpMode == MONITOR_FOCUS)
    // Clear focus flipflop and wait for trip
```

```
      {
      // we assume the autofocus flip flop threshold has been set
      // note that this loop should look at RETRACTDn, if it were used
      // clear the focus FF
      SetCtrl (Z_Control & (~RST_FOCUSn));
      SetCtrl (Z_Control | RST_FOCUSn);
      if ((ReadPort (STS_REG) & FOCUSDn) == 0)   // if trip immediately
        OpStatus = THRESH_TOO_LOW;
      else
        OpStatus = COMPARATOR_READY;
      post_events();
          // now hang out and wait for tripped
      while (((ReadPort (STS_REG) & FOCUSDn) != 0) &&
              (OpMode == MONITOR_FOCUS))
        {
        post_events();
        }
      if (OpMode == MONITOR_FOCUS)
        OpStatus = COMP_TRIPPED;
      else
        OpStatus = ABORTED;
      post_events();
      OpMode = NULL_MODE;
      }

    when (OpMode == RD_STATUS)
      // read status reg
      {
      OpStatus = ReadPort (STS_REG);
      OpMode = NULL_MODE;
          }

    when (OpMode == ABORT_CMD)
      // this shouldn't happen unless we are in another op mode
```

```
{
OpStatus = ABORTED;
OpMode = NULL_MODE;
}
□
```

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX C

M-2464-1P US

RECTIFIED SHEET (RULE 91)

```
module fastz_02
title 'Fastz ctrl pld
Chris Fairley Ultrapointe Corp. San Jose CA'

"       FASTZ_02.abl
"       Fast-Z axis logic
"       Ultrapointe Corp.
"       !!!! PROPRIETARY !!!!
"       FOR INTERNAL USE ONLY
"
"
"       Target Device:  GAL20RA10
"       Target Socket:  U29 of FAST Z-AXIS CTRLLR
"
"   Revised:
"


        fastz_02 device 'P20RA10';
        PLOADn          pin 1;
        CLK4MHZ         pin 2;
        R_ADCn          pin 3;
        ZSYNCn          pin 4;
        RST_ZSYNCn      pin 5;
        TRIPPEDn        pin 6;
        RST_FOCUSn      pin 7;
        READ_WRITEn     pin 8;
        ADDR_DATAn      pin 9;
        SYNC_MODE       pin 10;
        W_DACHIn        pin 11;
        OUT_En          pin 13;
        ADC_BSYn        pin 14;
        PKDET_MODE      pin 15;
        BUSY2n          pin 16;
        BUSY2n          istype  'reg_D, pos';
        RST_PKDET       pin 17;
        RST_PKDET       istype  'reg_D, pos';
        LD_DACn         pin 18;
        LD_DACn         istype  'com, pos';
        BUSY1n          pin 19;
        BUSY1n          istype  'reg_D, pos';
        XCVR_RWn        pin 20;
        XCVR_RWn        istype  'com, pos';
        CNVRTn          pin 21;
        CNVRTn          istype  'com, pos';
        ZSYNCDn         pin 22;
        ZSYNCDn         istype  'reg_D, pos';
        FOCUSDn         pin 23;
        FOCUSDn         istype  'reg_D, pos';

        H, L, X, C      = 1, 0, .X., .C.;
        RWn             = READ_WRITEn;
        ADn             = ADDR_DATAn;
        ZSn             = ZSYNCn;
        RZSn            = RST_ZSYNCn;
        SMOD            = SYNC_MODE;
        WDACn           = W_DACHIn;
        PKMOD           = PKDET_MODE;
        RADCn           = R_ADCn;
        ZSDn            = ZSYNCDn;
        LDACn           = LD_DACn;
        CVRTn           = CNVRTn;
        AA,BB,FF        = TRIPPEDn,RST_FOCUSn,PLOADn;
        CC,DD,EE        = H,H,H;

    equations
        " LD DAC on high byte write in async mode,
        " when ZSYNC happens in sync mode
        LD_DACn         = !( (SYNC_MODE & !ZSYNCDn) #
                           (!SYNC_MODE & !W_DACHIn) );
        LD_DACn.oe      = H;
```

```
" always writing in address mode, follow R/W in data mode
XCVR_RWn        = !((!ADDR_DATAn & !READ_WRITEn) #
                    ADDR_DATAn);
XCVR_RWn.oe     = H;

" ADC convert when read ADC in normal mode,
" assert and hold convert after ZSYNC happens
CNVRTn          = !( (!PKDET_MODE & !R_ADCn) #
                    (PKDET_MODE & !ZSYNCDn & RST_ZSYNCn) #
                    (PKDET_MODE & !CNVRTn & RST_ZSYNCn) );
CNVRTn.oe       = H;

" edge not required for FOCUSDn, so if already in focus
" after RST_FOCUSn, will set anyway without edge.
FOCUSDn         := H;
FOCUSDn.oe      = H;
FOCUSDn.clk     = H;
FOCUSDn.ap      = !TRIPPEDn;
FOCUSDn.ar      = !RST_FOCUSn;

" ZSYNCn edges are reliable and may want to reset ZSYNC
" while ZSYNCn is low (wide pulse), so use CLK input
ZSYNCDn         := L;
ZSYNCDn.oe      = H;
ZSYNCDn.clk     = !ZSYNCn;
ZSYNCDn.ap      = L;
ZSYNCDn.ar      = !RST_ZSYNCn;

" delayed versions of ADC_BSYn for pulse shaping
BUSY1n          := ADC_BSYn;
BUSY1n.oe       = H;
BUSY1n.clk      = CLK4MHZ;
BUSY1n.ap       = L;
BUSY1n.ar       = L;
```

```
          BUSY2n        := BUSY1n;
          BUSY2n.oe      = H;
          BUSY2n.clk     = CLK4MHZ;
          BUSY2n.ap      = L;
          BUSY2n.ar      = L;

          " reset peak detector as soon as conversion complete
          RST_PKDET       := ADC_BSYn & !BUSY2n;
          RST_PKDET.oe     = H;
          RST_PKDET.clk    = CLK4MHZ;
          RST_PKDET.ap     = L;
          RST_PKDET.ar     = L;


     " ZSYNCDn, FOCUSDn:
     test_vectors
      ([ FF, ZSYNCn, RST_ZSYNCn, TRIPPEDn, RST_FOCUSn ] -> [ ZSYNCDn, FOCUSDn ])
       [ EE,  H    ,    L      ,    H    ,     L      ] -> [   H    ,   H     ];
       [ EE,  H    ,    L      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  L    ,    H      ,    H    ,     H      ] -> [   L    ,   H     ];
       [ EE,  L    ,    H      ,    H    ,     H      ] -> [   L    ,   H     ];
       [ EE,  H    ,    H      ,    H    ,     H      ] -> [   L    ,   H     ];
       [ EE,  H    ,    L      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  L    ,    H      ,    H    ,     H      ] -> [   L    ,   H     ];
       [ EE,  H    ,    L      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    L    ,     H      ] -> [   H    ,   L     ];
       [ EE,  H    ,    H      ,    L    ,     H      ] -> [   H    ,   L     ];
       [ EE,  H    ,    H      ,    H    ,     H      ] -> [   H    ,   L     ];
       [ EE,  H    ,    H      ,    H    ,     L      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    H    ,     L      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    H    ,     H      ] -> [   H    ,   H     ];
       [ EE,  H    ,    H      ,    L    ,     H      ] -> [   H    ,   L     ];
       [ EE,  H    ,    H      ,    H    ,     L      ] -> [   H    ,   H     ];
```

```
" XCVR_RWn:
test_vectors
   ([ FF, AA,BB, ADn, RWn ] -> [XCVR_RWn])
    [ EE, CC,DD,  H ,  H  ] -> [   L    ];
    [ EE, CC,DD,  H ,  L  ] -> [   L    ];
    [ EE, CC,DD,  L ,  H  ] -> [   H    ];
    [ EE, CC,DD,  L ,  L  ] -> [   L    ];

" LD_DACn:
test_vectors
([FF,AA,BB, ZSn, RZSn, SMOD, WDACn, PKMOD, RADCn]-> [ZSDn, LDACn, CVRTn])
[ EE,CC,DD,  H ,    L ,    L ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    L ,     L ,   H  ]-> [  H ,    L ,   H  ];
[ EE,CC,DD,  L ,    H ,    L ,    H ,     L ,   H  ]-> [  L ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    L ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  H ,    L ,    H ,    L ,     L ,   H  ]-> [  H ,    H ,   H  ];

[ EE,CC,DD,  H ,    H ,    H ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
[ EE,CC,DD,  L ,    H ,    H ,    H ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  L ,    H ,    H ,    H ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  H ,    L ,    H ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    H ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
[ EE,CC,DD,  L ,    H ,    H ,    H ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  H ,    H ,    H ,    H ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   H  ]-> [  L ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    H ,    H ,     L ,   H  ]-> [  L ,    L ,   H  ];
[ EE,CC,DD,  H ,    L ,    H ,    H ,     L ,   H  ]-> [  H ,    H ,   H  ];
" CNVRTn:
"[FF,BB,ZSn, RZSn, SMOD, WDACn, PKMOD, RADCn ] -> [ ZSDn, LDACn, CVRTn ])
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   H  ] -> [ H ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   L  ] -> [ H ,    H ,   L  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   L  ] -> [ H ,    H ,   L  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     L ,   H  ] -> [ H ,    H ,   H  ];
[ EE,CC,DD,  H ,    H ,    L ,    H ,     H ,   H  ] -> [ H ,    H ,   H  ];
[ EE,CC,DD,  L ,    H ,    L ,    H ,     H ,   H  ] -> [ L ,    H ,   L  ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  H ,  H ] -> [ L ,  H ,  L ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  H ,  H ] -> [ L ,  H ,  L ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  L ,  H ] -> [ L ,  H ,  H ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  H ,  H ] -> [ L ,  H ,  L ];
[ EE,CC,DD, H ,  L ,  L ,  H ,  H ,  H ] -> [ H ,  H ,  H ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  H ,  H ] -> [ H ,  H ,  H ];
[ EE,CC,DD, H ,  H ,  L ,  H ,  H ,  H ] -> [ H ,  H ,  H ];

" RST_PKDET:
test_vectors
   ([ FF, AA,BB, CLK4MHZ, ADC_BSYn ] -> [ BUSY1n, BUSY2n, RST_PKDET ])
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];
    [ EE, CC,DD,    C    ,    H    ] -> [    H ,    H ,    L    ];

end fastz_02
```

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX D

M-2464-1P US

```
          ULTRAPOINTE CORPORATION
           SINGLE LEVEL EXPLOSION     C/N - 220  CHRISF  PAGE 1  RKPSTD01
      Format Type Product Structure Fields   Type of Components Input and Output
                                                    Alternate BOM
   Level-I0-Part  Description            QTY/Per QTY--Effective--UM   ITM
     Number                                      BAS   FROM   TO     #ON  Process-Op
                                                                     DRW
   0  000491      PCB,FAST-Z AXIS CNTRLR ASSY                      EA

    1  000061   CONN, RJ-45 PHONE JACK      2.   42393 999999 EA    0
                 J4        J5
    1  000066   DIODE, SA5.0A, TRANSORB     1.   70192 999999 EA    72
                 D12
    1  000067   DIODE, LED, RED, T1-1/4     2.   50193 999999 EA    0
                 D1        D11
    1  000068   CAP, 0.047 UF, 50V, CER, AXIAL36.   42493 999999 EA    0
                 C11       C12       C13       C16       C18       C19
                 C2        C21       C24       C26       C29       C3
                 C30       C31       C32       C33       C35       C36
                 C37       C38       C39       C4        C40       C41
                 C42       C43       C44       C45       C46       C49
                 C5        C52       C53       C54       C57       C9
    1  000069   CAP, 470 PF, 50V, CER, AXIAL 1.   70192 999999 EA    17
                 C6
    1  000073   CONN, SOC, 18 PIN, FEM      1.   70192 999999 EA    27
                 J6
    1  000074   CONN, SOC, 6 PIN, FEM       1.   70192 999999 EA    70
                 J3
    1  000078   CAP, 33 UF, 16V, LYTIC, RAD 3.   70192 999999 EA    42
                 C50       C55       C56
    1  000079   SOC, 24 PIN, DIP, 0.3" CTRS  1.   70192 999999 EA    71
                 (U26)
    1  000101   IC, DS1233 ECONO RESET      1.   70192 999999 EA    33
                 U2
    1  000105   IC, LM7805, REG, 5V         1.   70192 999999 EA    38
                 U27
          ULTRAPOINTE CORPORATION
           SINGLE LEVEL EXPLOSION      C/N - 220   CHRISF   PAGE 2  RKPSTD01
     Format Type Product Structure Fields      Type of Components Input and Output
                                                      Alternate BOM
   Level-I0-Part   Description             QTY/Per QTY--Effective--UM     ITM
     Number                                        BAS    FROM   TO       #ON  Process-Op
                                                                          DRW
   0  000491    PCB,FAST-Z AXIS CNTRLR ASSY                           EA
    1  000148    IC,74ALS245,OCTAL BUS XCVR 1.  70192 999999 EA       3
                  U17
    1  000152    IC,74ALS574,OCTAL FLIP-FLOP 3.  70192 999999 EA      7
                  U12         U19         U7
    1  000165    RES, 75 OHM, 1%, 1/4W,MF   1.  70192 999999 EA     60
                  R37
    1  000166    RES, 100 OHM, 1%, 1/4W,MF 6.  42493 999999 EA      0
                  R16         R18         R25         R38         R39         R6
    1  000168    RES, 261 OHM, 1%, 1/4W,MF 2.  70192 999999 EA     44
                  R3          R43
    1  000171    RES, 20K OHM, 1%, 1/4W,MF 1.  70192 999999 EA     61
                  R29
    1  000172    RES, 10K OHM, 1%, 1/4W,MF 8.  70192 999999 EA     48
                  R1          R10         R11         R13         R15         R28
                  R36         R9
    1  000173    RES, 5.11K OHM, 1%, 1/4W,MF 1.  70192 999999 EA    62
                  R30
    1  000174    RES, 1K OHM, 1%, 1/4W,MF   4.  70192 999999 EA     45
                  R19         R35         R41         R42
    1  000176    IC, AD712JN, DUAL OP AMP   3.  70192 999999 EA     13
                  U11         U22         U6
    1  000179    CONN,SMB,JACK,PCB,RT ANGLE,75 2.  70192 999999 EA     66
                  J7          J8
    1  000194    CAP,1000PF,50V,CER,AXIAL   2.  70192 999999 EA     18
                  C15         C48
    1  000197    CAP,33PF,100V,CER,AXIAL    2.  70192 999999 EA     25
                  C10         C28
    1  000199    TESTPOINT,PCB,STR,YEL      12.  70192 999999 EA     67
```

```
           ULTRAPOINTE CORPORATION
           SINGLE LEVEL EXPLOSION     C/N - 220  CHRISF  PAGE 3  RKPSTD01
      Format Type Product Structure Fields   Type of Components Input and Output
                                                  Alternate BOM
  Level-I0-Part  Description          QTY/Per QTY--Effective--UM    ITM
   Number                                     BAS   FROM    TO      #ON  Process-Op
                                                                    DRW
  0  000491   PCB,FAST-Z AXIS CNTRLR ASSY                       EA
              TP1      TP10      TP11      TP12      TP2      TP3
              TP4      TP5       TP6       TP7       TP8      TP9
   1 000246   SW SPST PCB NO MOM 6 MM 2.  70192 999999 EA    43
              S1       S2
   1 000279   MODULE,LON XCVR,TP RS-485 1.  70192 999999 EA    68 M1
   1 000281   DIODE, 1N5711, SCHOTTKY  6.  70192 999999 EA    9
              D10      D3        D4        D6        D7       D9
   1 000283   CAP,1UF,50V,CER,AXIAL    1.  70192 999999 EA    26
              C25
   1 000289   IC,AD843,OP-AMP,FET INPUT 1.  70192 999999 EA    16
              U14
   1 000292   IC,LM336-2.5,VOLTAGE REF 1.  42393 999999 EA    0
              U1
   1 000295   CAP,6.8UF,20V,TANT,RADIAL 2.  70192 999999 EA    41
              C17      C51
   1 000300   RES,1M OHM,1%,1/4W,MF    2.  42393 999999 EA    0
              R14      R27
   1 000302   RES,100K OHM,1%,1/4W,MF 2.  70192 999999 EA    46
              R26      R40
   1 000305   RES,12.1K OHM,1%,1/4W,MF 1.  42393 999999 EA    0
              R17
   1 000355   SCR,PAN,PHL,SS,6-32X1/4,SEM 6.  50193 999999 EA    0
              (M1)
   1 000400   CONN,3 PIN,HDR,PCB,RT ANGLE 1.  70192 999999 EA    28
              J1
   1 000401   CONN,4 PIN,HDR,PCB,RT ANGLE 1.  70192 999999 EA    29
              J10
   1 000402   CONN,5 PIN,HDR,PCB,RT ANGLE 1.  70192 999999 EA    30
```

```
ULTRAPOINTE CORPORATION
SINGLE LEVEL EXPLOSION      C/N - 220   CHRISF   PAGE 4  RKPSTD01
Format Type Product Structure Fields      Type of Components Input and Output
                                          Alternate BOM
Level-I0-Part  Description            QTY/Per QTY--Effective--UM     ITM
  Number                                      BAS   FROM   TO        #ON  Process-Op
                                                                     DRW
0  000491     PCB,FAST-Z AXIS CNTRLR ASSY                         EA
              J2
1  000491SD PCB,FAST-Z AXIS CNTRLR SCHEM 0.  50193 999999 EA       0
1  000493     PCB,FAST-Z AXIS CNTRLR FAB 1.   70192 999999 EA    74
1  000495     IC,PA85,POWER OP-AMP       1.   70192 999999 EA    40
              U8
1  000500     IC,AD7575,8-BIT ADC,5 USEC 1.   70192 999999 EA    15
              U9 1  000501             IC,AD7541,12-BIT DAC 1.   70192
999999 EA     14
              U4
1  000502     IC,DAC0808,8-BIT DAC       1.   70192 999999 EA    31 U15
1  000504     IC,DG211,QUAD ANALOG SWITCH 1.   70192 999999 EA     32
              U13
1  000506     CONN,8 PIN,HDR,PCB,STR,MOLEX 1.   70192 999999 EA     39
              J9
1  000507     CAP,68PF,200V,5%,CER,RAD 2.   22294 999999 EA     0
              C14         C7
1  000509     CAP,0.1UF,200V,20%,CER,RAD 2.   70192 999999 EA     21
              C1          C34
1  000537     IC,74LS14,HEX SCHMITT TRGR INV 1.   70192 999999 EA     1
              U18
1  000538     IC,74ALS138,3-8 DECODER    1.   70192 999999 EA     2
              U25
1  000539     IC,74ALS273,OCTAL FLIP-FLOP 3.   70192 999999 EA     4
              U10         U24         U5
1  000540     IC,74ALS541,OCTAL BUFFER  1.   70192 999999 EA     5
              U20
1  000541     IC,74ALS573,OCTAL XPRNT LATCH 1.   70192 999999 EA     6
              U16
```

```
          ULTRAPOINTE CORPORATION
           SINGLE LEVEL EXPLOSION     C/N - 220  CHRISF  PAGE 5  RKPSTD01
     Format Type Product Structure Fields    Type of Components Input and Output
                                                     Alternate BOM
   Level-IO-Part  Description              QTY/Per QTY--Effective--UM    ITM
     Number                                    BAS    FROM     TO        #ON  Process-Op
                                                                         DRW
   0  000491   PCB.FAST-Z AXIS CNTRLR ASSY                          EA
    1  000542   IC,74ALS139,DUAL 2-4 DECODER 1.  70192 999999 EA       8
                U23
    1  000544   IC,AD589,1.23V REF           1.  70192 999999 EA      12
                D2
    1  000545   CAP,100PF,100V,CER,AXIAL     1.  70192 999999 EA      24
                C23
    1  000546   IC,LM311,COMPARATOR          1.  70192 999999 EA      36
                U21
    1  000547   RES,0 OHM,JUMPER             2. 120193 999999 EA       0
                R4          R7
    1  000548   RES,16.2K OHM,1%,1/4W,MF 1.      70192 999999 EA      49
                R21
    1  000549   RES,1.62K OHM,1%,1/4W,MF 1.      70192 999999 EA      50
                R12
    1  000550   RES,68.1K OHM,1%,1/4W,MF 1.      70192 999999 EA      51
                R24
    1  000551   RES,162K OHM,1%,1/4W,MF  1.      70192 999999 EA      53
                R20
    1  000552   RES,3.83K OHM,1%,1/4W,MF 1.      70192 999999 EA      54
                R23
    1  000554   RES,10.0 OHM,1%,1/4W,MF  1.      70192 999999 EA      56
                R2
    1  000555   RES,8.25 OHM,1%,1/4W,MF  1.      70192 999999 EA      57
                R5
    1  000556   RES,2.49K OHM,1%,1/4W,MF 2.      70192 999999 EA      58
                R31         R32
    1  000557   RES,1.21K OHM,1%,1/4W,MF 2.      70192 999999 EA      63
```

ULTRAPOINTE CORPORATION

SINGLE LEVEL EXPLOSION C/N - 220 CHRISF PAGE 6 RKPSTD01

Format Type Product Structure Fields Type of Components Input and Output

Alternate BOM

Level-I0-Part	Number	Description	QTY/Per BAS	QTY--Effective--FROM	TO	UM	ITM #ON DRW	Process-Op
0	000491	PCB,FAST-Z AXIS CNTRLR ASSY				EA		
		R33 R34						
1	000562	CAP,0.1UF,50V,POLY,RAD	2.	70192	999999	EA	19	
		C22 C27						
1	000653	NUT,HEX,SS,6-32,W/LOCK WSHR	2.	42393	999999	EA	0	
1	000728	IC,OP-177GP,OP-AMP	1.	42393	999999	EA	0	
		U3						
1	000760	RES,49.9K,OHM,1%,1/4W,MF	1.	42393	999999	EA	0	
		R22						
1	000803	SCR,BUT,SOC,SS,6-32X3/8	2.	42393	999999	EA	0	
1	000868	DIODE,1N5235A,ZENER,6.8V	1.	42393	999999	EA	0	
		D5						
1	000872	RES,562 OHM,1%,1/4W,MF	1.	42393	999999	EA	0	
		R44						
1	000874	HEATSINK,TO-3 XSTR	1.	42393	999999	EA	0	
		(U8)						
1	006008	RES,POT,10K OHM,1T,3/8 SQ	1.	22294	999999	EA	0	
1	011012	DIODE,SA150,TRANSIENT SUPPRESS	1.	80193	999999	EA	0	
		D8						
1	012022	STANDOFF HEX AL F/F,6-32X1/2	3.	50193	999999	EA	0	
		(M1)						
1	012048	CONN,2 PIN SHUNT JUMPER	1.	120193	999999	EA	0	
		(JP1)						
1	012057	CONN,2 PIN HDR, STRAIGHT	1.	120193	999999	EA	0	
		JP1						

ULTRAPOINTE CORPORATION
SINGLE LEVEL EXPLOSION C/N - 220 CHRISF PAGE 1 RKPSTD01
Format Type Product Structure Fields Type of Components Input and Output
Alternate BOM

Level-I0-Part Number	Description	QTY/Per	QTY–Effective–UM			ITM #ON DRW	Process-Op
		BAS	FROM	TO			
0 000625	ASSY,X-Y-Z-AF STAGE				EA		
1 000250	STAGE SYSTEM X Y Z	1.	70192	999999	EA	1	
1 000253	PZT MTG AFSTAGE	1.	70192	999999	EA	5	
1 000254	FLEXURE AFSTAGE	16.	70192	999999	EA	14	
1 000256	PZT RETAINER AFSTAGE	1.	70192	999999	EA	6	
1 000257	ROTATING BAR AFSTAGE	4.	70192	999999	EA	10	
1 000258	FLEXURE CLAMP AFSTAGE	32.	70192	999999	EA	15	
1 000259	PRELOAD SCREW AFSTAGE	2.	70192	999999	EA	20	
1 000260	STATIONARY BAR AFSTAGE	4.	70192	999999	EA	9	
1 000261	TOP PLATE AFSTAGE	1.	70192	999999	EA	11	
1 000263	PZT 30 MICRON 100VDC	1.	70192	999999	EA	7	
1 000264	COVER AFSTAGE	1.	70192	999999	EA	12	
1 000266	VACUUM PLATE,AF STAGE	1.	70192	999999	EA	13	
1 000344	SCR,CAP,SOC,SS,10-32X5/8	10.	70192	999999	EA	33	
1 000357	SCR,CAP,SOC,SS,6-32X3/8	2.	31894	999999	EA	0	
1 000418	SENSOR,PROX,50-150UM,KAMAN	1.	70192	999999	EA	3	
1 000516	PLATE,ZSTAGE ADAPTOR	1.	70192	999999	EA	2	
1 000530	WAFER CHUCK	1.	70192	999999	EA	18	
1 000581	SCR,CAP,SOC,SS,10-32 X 3/8	4.	31894	999999	EA	0	
1 000614	SCR,PAN,SLT,SS,8-32X1/2,W/SEAL	4.	70192	999999	EA	19	
1 000780	HOSE BARB,,10-32 TO 1/16 HOSE	3.	50193	999999	EA	0	
1 000781	SOLENOID VALVE,5 PORT,BASE MTG	3.	90193	999999	EA	0	
1 000784	HOSE,1/16ID X 1/8OD,8X,URTHN	3.	50193	999999	EA	0	
1 000798	SCR,BUT,SOC,SS,4-40X3/8	13.	70192	999999	EA	34	

ULTRAPOINTE CORPORATION
SINGLE LEVEL EXPLOSION C/N - 220 CHRISF PAGE 2 RKPSTD01
Format Type Product Structure Fields Type of Components Input and Output
Alternate BOM

Level-I0-Part Number		Description	QTY/Per	QTY--Effective--UM				ITM #ON DRW	Process-Op
				BAS	FROM	TO			
0	000625	ASSY,X-Y-Z-AF STAGE					EA		
1	000799	SCR,SET,SOC,SS,4-40X9/32,NYTIP	1.		70192	999999	EA	35	
1	000805	SCR,CAP,SOC,SS,8-32X3/8	4.		70192	999999	EA	28	
1	000813	SCR,CAP,SOC,SS,10-32X3/4	4.		31894	999999	EA	0	
1	000853	BALL,1/8 DIA,SS	3.		70192	999999	EA	8	
1	000854	SCR,FLT,SOC,SS,6-32X1/2	64.		31894	999999	EA	0	
1	000858	SCR,SET,SOC,SS,8-32X3/4,CUP PT	4.		31894	999999	EA	0	
1	000860	0-RING,1/16WX1/160DX11/640D NO	3.		70192	999999	EA	24	
1	012050	SCR,FLT HD, PHLPS,SS,#4X0.5 LG	4.		90193	999999	EA	0	
1	012058	CONN,AC PWR INLET,250VAC,10 A	1.		11894	999999	EA	0	

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX E

M-2464-1P US

RECTIFIED SHEET (RULE 91)

```
/**********************************************
 * function description:
 *
 * Some functions are not listed
 * These functions are described here
 **********************************************
 */
```

cfg_GetLockRdConfig	Get the pointer to system configuration data structure
da_GlobalLonPointer	Get the pointer to system status data structure
lon_APIQuery	Query value of a specified hardware sybsystem parameter
lon_APICmd	Set a specified hardware sybsystem parameter to specified value
lonui_SetFastZPos	Set the fine Z stage to specified position
lonuiLocalFineAF	Do fine stage auto-focus
util_sginap	Causes the program to sleep(wait) for specified number of 10 milli-seconds.
XtMalloc	Allocated system memory
XtFree	Release previousely allocated memory.
longmi_UpdateNV	Update specified entry in hardware status table

Ionimg_QueryNetVar

Query and get specified value(8bit) of hardware subsystem parameter.

Ionimg_QueryNetVar16

Query and get specified value(16 bit) of hardware subsystem parameter.

LonTimeBlockReadOneAndDecodeMsg

Wait for a message from hardware subsystem, when the message is received, decode it to get the message's subsystem parameter ID, and its value.

```
/*************************************************************
 * (c) Copyright 1992,1993,1994 Ultrapointe Corp.
 * All rights reserved
 *
 * Auto focus
 *
*************************************************************
*/

#define WORKING_DIST_UM      800
#define WORKING_DIST_10X_UM        1600
#define BACKUP_BEFORE_AF_UM        2000       /* if closer than this backdown first */

#define AF2_INTEN_DEC                    .85       /* if PMT reset, lower by this amount */
#define AF2_INTEN_DEC_NORMAL       .95
#define PMT_CLAMPED            1              /* pmt latched overload */

static UI_LON_IMAGE      UIimage;

#define LASER_I_PERCENT_CHG_PWR        70    /* above this intensity, change power  */
#define MAX_LASER_PWR                        125
#define MIN_PMT_GAIN                         245
#define MAX_PMT_ZERO                         132
#define MIN_PMT_ZERO             70
                     /* laser pwr: 0-255 = 0mw-50mw */
```

```
#define MAX_LASER_PWR              125


#define NUM_NV_IN_NODE             50
#define GENERIC_OPMODE_NV_ADDR         2
#define GENERIC_OPSTATUS_NV_ADDR       3

#define OPMODE_TIMEOUT_MS              8000
#define OPMODE_TIMEOUT_TICK            (OPMODE_TIMEOUT_MS * CLK_TCK /1000)
#define AF_READ_MS                 20
#define AF_READ_TICK               (AF_READ_MS * CLK_TCK /1000)
#define VOL_CHK_LOOP_TIME          30     /* 10ms */
#define C_AF_COMPARATOR_RDY        8
#define C_AF_THRES_TOO_LO   9
#define C_AF_COMP_TRIP             0xa
#define C_AF_ABORT             0xb

#define PMT_CLAMPED   1             /* pmt latched overload */
#define LASER_PWR_SLIDER_RANGE         100

typedef struct CALLBK_STRUCT_tag
{
  void (*CallBkPtr)(void *, int, int, int); /* FctPtr to callback           */
  void *ParmPtr ;               /* ParameterPtr for callback              */
  void *NextCBPtr ;
} CALLBK_STRUCT ;

typedef struct HARDWR_IMAGE_tag
 {
 char          *NVName;
 int            iNVValue;
 char           chArray[LON_ARRAY_DATA_SZ];     /* for FastZ intensity array
     */
 int            iNumByte;        /* 1 or 2 or 16byte NV               */
 CALLBK_STRUCT  *CBStructPtr;           /* start of list callback struct      */
 NETVAR_DIRECT  NVDirection;            /*                 */
 BOOL                 UpdatedFlg;
```

```
  } HARDWR_IMAGE_ENTRY ;

typedef struct LON_NODE_IMAGE_tag
  {
  char                 *NodeName;
  HARDWR_IMAGE_ENTRY   NVImg[ NUM_NV_IN_NODE];
  } LON_NODE_IMAGE;


/****************************************************************
 * struct for the hardware image table
 * items in the table are sequential, keyed to the
 * nodeID and NV number
 *
 ****************************************************************
 */
static LON_NODE_IMAGE        HardWrImgTable[] = {
{ "dummy" ,
{                              /* 0, node 0 is not used in LON-talk       */
      { "dummy", 0,{0}, -1,NULL, FALSE, FALSE, },
} },

{ "LaserPwrControl",
{                              /* 1, Laser power controller */
/* 0*/  { "SoftWr Ver",     0,{0}, 1, NULL, eNetVarToSGI,  FALSE },  /*O Laser SW Rev,
      */
/* 1*/  { "ResetStatus",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },  /*O Laser ResetStat,     */
/* 2*/  { "ModeCmd",       0,{0}, 1, NULL, eNetVarFromSGI,FALSE },   /*  Laser Mode,
cmd           */
/* 3*/  { "LaserStatus",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },  /*O Laser Status,     */
/* 4*/  { "LaserOn/Off",HDWR_LASER_OFF,     {0}, 1, NULL, eNetVarFromSGI,FALSE },
/*  Laser On/Off  = 1/0, cmd       */
/* 5*/  { "Run/Idle",        HDWR_LASER_IDLE,    {0}, 1, NULL, eNetVarFromSGI,FALSE
},  /*  Laser Run/Idle = 1/0,cmd           */
/* 6*/  { "Light/Curr",      HDWR_LASER_LIGHT,   {0}, 1, NULL, eNetVarFromSGI,FALSE
},  /*  Laser ModeSel,       cmd  light/current  1/0 */
/* 7*/  { "CurrentCmd",    HDWR_DEF_LSR_CUR,  {0}, 1, NULL, eNetVarFromSGI,FALSE
}, /* Laser Current,       cmd  0-255     */
/* 8*/  { "PowerCmd",      HDWR_DEF_LSR_PWR, {0}, 1, NULL, eNetVarFromSGI,FALSE
}, /* Laser Power,         cmd  0-255     */
/* 9*/  { "WhiteLtPwr",    HDWR_DEF_LSR_WL_PWR,{0}, 1, NULL, eNetVarFromSGI,
FALSE }, /* WhiteL Pwr,    cmd  0-255       */
/*10*/  { "SpareCmd",      0,{0}, 1, NULL, eNetVarFromSGI,FALSE },   /* SpareCmd cmd
0-255  */
/*11*/  { "InterLock",     0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O LaserInterlock
open/close= ??????? */
/*12*/  { "CurrCheck",     0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O
LaserCurrCheck DAC          0-255     */
/*13*/  { "PwrCheck",      0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O LaserPwrCheck
DAC       0-255     */
/*14*/  { "WLPwrCheck",  0,{0}, 1, NULL, eNetVarToSGI,  FALSE },     /*O
WhiteLpwrCheck DAC          0-255     */
/*15*/  { "SpareCheck",    0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O SpareCheck
DAC       0-255     */
/*16*/  { "HeadTempMon",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },      /*O
LaserHeadTempMonitor      0-255 */
/*17*/  { "SpareTemp",     0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O SpareTemp
0-255  */
/*18*/  { "LaserPwrMon",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },      /*O
LaserPwrMonitor          0-255     */
/*19*/  { "CurrMonitor",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },      /*O
laserCurrMonitor         0-255     */
        { NULL,            0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "CameraFilterWheel",
{                               /* 2, camera filter wheel */
/* 0*/  { "SoftWr Ver",    0,{0}, 1, NULL, eNetVarToSGI,  FALSE },   /*O SW Rev,*/
/* 1*/  { "ResetStatus",0,{0}, 1, NULL, eNetVarToSGI,  FALSE },      /*O ResetStat,    */
/* 2*/  { "ModeCmd",       0,{0}, 1, NULL, eNetVarFromSGI,FALSE },   /* Mode, cmd    */
/* 3*/  { "Status",   0,{0}, 1, NULL, eNetVarToSGI,  FALSE },        /*O Status,    */
/* 4*/ { "PositionCmd",HDWR_LSR_CAM_APASS,{0}, 1, NULL, eNetVarFromSGI,FALSE
},             /* wheel position      1-6        */
/* 5*/ { "HomeStatus",     0,{0}, 1, NULL, eNetVarToSGI,  FALSE },        /*O home
sensor status on/off= ????  */
```

```
/* 6*/ { "PosStatus",        1,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O wheel
pos status        1-6                 */
/* 7*/ { "StepCnt",  1,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O Step Count
???     */
/* 8*/ { "StepDelay",600,{0},  2, NULL, eNetVarFromSGI,FALSE },            /*O Step
delay      ???          */
        { NULL,              0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "LaserLineFilter",
{                          /* 3, laser line filter wheel */
/* 0*/  { "SoftWr Ver",    0,{0},  1, NULL, eNetVarToSGI,  FALSE },    /*O SW Rev,     */
/* 1*/  { "ResetStatus",0,{0},  1, NULL, eNetVarToSGI,  FALSE },        /*O ResetStat,     */

/* 2*/  { "ModeCmd",     0,{0},  1, NULL, eNetVarFromSGI,FALSE },           /*  Mode,
   cmd        */
/* 3*/  { "Status",   0,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O Status,    */
/* 4*/  { "PositionCmd",HDWR_LSR_LINE_ALL,{0},  1, NULL, eNetVarFromSGI,FALSE },/*
wheel position            1-6                 */
/* 5*/  { "HomeStatus",     0,{0},  1, NULL, eNetVarToSGI,  FALSE },    /*O home sensor
status on/off= ????  */
/* 6*/  { "PosStatus",        1,{0},  1, NULL, eNetVarToSGI,  FALSE },    /*O wheel pos status
  1-6   */
/* 7*/  { "StepCnt",  1,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O Step Count
???     */
/* 8*/  { "StepDelay",600,{0},  2, NULL, eNetVarFromSGI,FALSE },            /*O Step
delay       ???          */
        { NULL,              0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "LaserAttenFilter",
{                          /* 4, laser attenuator filter wheel */
/* 0*/  { "SoftWr Ver",    0,{0},  1, NULL, eNetVarToSGI,  FALSE },    /*O SW Rev,     */
/* 1*/  { "ResetStatus",0,{0},  1, NULL, eNetVarToSGI,  FALSE },        /*O ResetStat,     */
/* 2*/  { "ModeCmd",     0,{0},  1, NULL, eNetVarFromSGI,FALSE },   /*  Mode,   cmd  */
/* 3*/  { "Status",   0,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O Status,    */
/* 4*/ { "PositionCmd",HDWR_LSR_ATTEN_HIG,{0},  1, NULL, eNetVarFromSGI,FALSE
},     /*  wheel position        1-6           */
/* 5*/ { "HomeStatus",     0,{0},  1, NULL, eNetVarToSGI,  FALSE },    /*O home sensor
```

```
status on/off= ???? */
/* 6*/ { "PosStatus",        1,{0}, 1, NULL, eNetVarToSGI, FALSE },    /*O wheel pos status
  1-6  */
/* 7*/ { "StepCnt", 1,{0}, 1, NULL, eNetVarToSGI, FALSE },    /*O Step Count      ??? */
/* 8*/ { "StepDelay",600,{0}, 2, NULL, eNetVarFromSGI,FALSE },    /*O Step delay???  */
       { NULL,            0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "Bright/DarkField",
{                             /* 5, bright/dark field filter wheel */
/* 0*/  { "SoftWr Ver",    0,{0}, 1, NULL, eNetVarToSGI, FALSE },    /*O SW Rev,      */
/* 1*/  { "ResetStatus",0,{0}, 1, NULL, eNetVarToSGI, FALSE },       /*O ResetStat,    */
/* 2*/  { "ModeCmd",      0,{0}, 1, NULL, eNetVarFromSGI,FALSE },  /* Mode, cmd   */
/* 3*/  { "Status",   0,{0}, 1, NULL, eNetVarToSGI, FALSE },         /*O Status,   */
/* 4*/ { "PositionCmd",HDWR_BRDRK_FLD_BRI,{0}, 1, NULL, eNetVarFromSGI,FALSE
},    /*  pos select bright/dark= ???     */
/* 5*/ { "PosStatus",     0,{0}, 1, NULL, eNetVarToSGI, FALSE },      /*  pos status
bright/dark= ???    */
/* 6*/ { "Sw1Status",     0,{0}, 1, NULL, eNetVarToSGI, FALSE },      /*O sw1 status
on/off = ???          */
/* 7*/ { "Sw2Status",       1,{0}, 1, NULL, eNetVarToSGI, FALSE },    /*O sw2 statuson/off
= ???    */
/* 8*/ { "StepCnt", 1,{0}, 1, NULL, eNetVarToSGI, FALSE },    /*O Step Count      ??? */
       { NULL,     0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "Page Scanner",
{                            /* 6, page scanner   */
/* 0*/  { "SoftWr Ver",    0,{0},    1, NULL, eNetVarToSGI,  FALSE },     /*O SW Rev,*/
/* 1*/  { "ResetStatus",0,{0},    1, NULL, eNetVarToSGI,  FALSE },     /*O ResetStat,    */
/* 2*/  { "ModeCmd",     0,{0},    1, NULL, eNetVarFromSGI, FALSE },   /* Mode, cmd
        */
/* 3*/  { "Status",  0,{0},    1, NULL, eNetVarToSGI,  FALSE },      /*O Status,   */
/* 4*/ { "VertSync",       HDWR_DEF_PAGE_VERT,  {0}, 2, NULL, eNetVarFromSGI,
FALSE },   /*  vertSync          0-65536     */
/* 5*/ { "TraceTiming",HDWR_DEF_PAGE_TRACE,  {0}, 2, NULL, eNetVarFromSGI,
FALSE },   /*  TraceTiming              0-65536     */
/* 6*/ { "RetraceTime",HDWR_DEF_PAGE_RETRACE,{0}, 2, NULL, eNetVarFromSGI,
FALSE },   /*  RetraceTiming     0-65536     */
```

```
/* 7*/ { "StartAmpOff".HDWR_DEF_PAGE_OFFSET. {0}. 2. NULL. eNetVarFromSGI.
FALSE },    /* StartAmplitude-offset 0-65536  */
/* 8*/ { "Increment",       HDWR_DEF_PAGE_INCREM. {0}. 2. NULL. eNetVarFromSGI.
FALSE },    /* Incr Amplitude    0-65536      */
/* 9*/ { "Decrement",       HDWR_DEF_PAGE_DECREM. {0}. 2. NULL. eNetVarFromSGI.
FALSE },    /* Decr Amplitude    0-65536      */
/*10*/          { "CtlRegValue".HDWR_DEF_PAGE_CTRL_REG.{0}.1. NULL.
eNetVarFromSGI. FALSE },       /* ControlRegValue          0-255          */
/*11*/          { "XAxisAmpl",      HDWR_DEF_PAGE_XAMP.   {0}. 2. NULL.
eNetVarFromSGI. FALSE },       /* X-axis Amplitude         0-4095      */
/*12*/          { "YAxisAmpl",      HDWR_DEF_PAGE_YAMP.   {0}. 2. NULL.
eNetVarFromSGI. FALSE },       /* Y-axis Amplitude         0-4095      */
/*13*/          { "XAxisPhase",     HDWR_DEF_PAGE_XPHASE. {0}. 2. NULL.
eNetVarFromSGI. FALSE },       /* X-axisPhase              0-4095      */
       { NULL,              0,{0}.   -1. NULL. eNetVarFromSGI. FALSE },
} }.
{ "FastZ Ctrl",
{                              /* 7. fast-z-axis-controller   */
/* 0*/ { "SoftWr Ver".    0,{0}.  1. NULL. eNetVarToSGI.   FALSE }.   /*O SW Rev,     */
/* 1*/ { "ResetStatus",0,{0}.  1. NULL. eNetVarToSGI.   FALSE }.        /*O ResetStat,    */
/* 2*/ { "OpMode",         0,{0}.  1. NULL. eNetVarFromSGI. FALSE },           /* OpMode,
     cmd      */
/* 3*/ { "OpStatus",       0,{0}.  1. NULL. eNetVarToSGI.   FALSE },   /*O OpStatus,  */
/* 4*/ { "OpParArr1",      0,{0}.  1. NULL. eNetVarFromSGI.FALSE },   /*  cmd,       */
/* 5*/ { "OpParArr2",      0,{0}.  1. NULL. eNetVarFromSGI.FALSE },   /*  cmd,       */
/* 6*/ { "OpStsArr1",      0,{0}.  1. NULL. eNetVarToSGI.  FALSE },    /*O Status,    */
/* 7*/ { "OpStsArr2",      0,{0}.  1. NULL. eNetVarToSGI.  FALSE },    /*O Status,    */
/* 8*/ { "AFocusThresh".HDWR_DEF_Z_AFTHRESH.{0}.1. NULL. eNetVarFromSGI.
FALSE },          /* AFocusThresh. cmd     */
/* 9*/ { "ZPositionCmd",  HDWR_DEF_ZPOSCMD.{0}.2. NULL. eNetVarFromSGI. FALSE
},      /* ZPosCmd.      cmd       */
/*10*/ { "ZControl",          HDWR_DEF_ZCONTROL.{0}.1. NULL. eNetVarFromSGI.
FALSE },     /* ZControl.     cmd        */
/*11*/ { "MuxSelect",          HDWR_DEF_ZMUXSEL.{0}.1. NULL. eNetVarFromSGI.
FALSE },     /* MuxSel.  cmd      */
/*12*/ { "ZStep",         HDWR_DEF_ZSTEP.{0}.1. NULL. eNetVarFromSGI. FALSE },/*
ZStep.     cmd   */
```

```
/*13*/ { "NumFrames",  HDWR_DEF_ZNUMFRAME,{0},1, NULL, eNetVarFromSGI,
FALSE },   /* NumFrames,   cmd   */
/*14*/ { "ZStatus", 0,{0},  1, NULL, eNetVarToSGI,  FALSE },   /*O ZStatus,   */
/*15*/ { "PosStatus",  0,{0},  2, NULL, eNetVarToSGI,  FALSE },       /*O PositionStatus
*/
/*16*/ { "Intensity0", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*17*/ { "Intensity1", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*18*/ { "Intensity2", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*19*/ { "Intensity3", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*20*/ { "Intensity4", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*21*/ { "Intensity5", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*22*/ { "Intensity6", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*23*/ { "Intensity7", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*24*/ { "Intensity8", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*25*/ { "Intensity9", 0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*26*/ { "Intensity10",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*27*/ { "Intensity11",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*28*/ { "Intensity12",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*29*/ { "Intensity13",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*30*/ { "Intensity14",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*31*/ { "Intensity15",0,{0}, 16, NULL, eNetVarToSGI,  FALSE },
/*32*/ { "SelfTest",   FASTZ_OPCMD_SELFTEST, {0},1, NULL, eOpModeCmd, FALSE },
/* SelfTest,    cmd  */
/*33*/ { "AutoFocus",      FASTZ_OPCMD_AUTOFOCUS,{0},1, NULL, eOpModeCmd,
FALSE }, /* AutoFocus,   cmd  */
/*34*/ { "ReadVideo",      FASTZ_OPCMD_READVIDEO,{0},1, NULL, eOpModeCmd,
FALSE }, /* ReadVideo,   cmd  */
/*35*/ { "ReadFeedBk",     FASTZ_OPCMD_READFDBK, {0},1, NULL, eOpModeCmd,
FALSE }, /* ReadFeedBk,  cmd  */
/*36*/ { "ReadHV",     FASTZ_OPCMD_READHV,  {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadHV,      cmd  */
/*37*/ { "SetForVol",      FASTZ_OPCMD_SETFORVOL,{0},1, NULL, eOpModeCmd,
FALSE }, /* SetForVol,  cmd   */
/*38*/ { "MonitorFoc",     FASTZ_OPCMD_MONITORFOC,{0},1,NULL, eOpModeCmd,
FALSE }, /* monitor focus,cmd  */
/*39*/ { "Abort",   FASTZ_OPCMD_ABORT,    {0},1, NULL, eOpModeCmd, FALSE }, /*
Abort,       cmd   */
```

```
/*40*/ { "ReadStatus",     FASTZ_OPCMD_RDSTATUS, {0},1, NULL, eOpModeCmd,
FALSE }, /* ReadStatus,  cmd   */
/*41*/ { "AutoFocus0",  FASTZ_OPCMD_FAST_AF0, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*42*/ { "AutoFocus1",  FASTZ_OPCMD_FAST_AF1, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*43*/ { "AutoFocus2",  FASTZ_OPCMD_FAST_AF2, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*44*/ { "AutoFocus3",  FASTZ_OPCMD_FAST_AF3, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*45*/ { "AutoFocus4",  FASTZ_OPCMD_FAST_AF4, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*46*/ { "AutoFocus5",  FASTZ_OPCMD_FAST_AF5, {0},1, NULL, eOpModeCmd, FALSE
}, /* ReadStatus,  cmd    */
/*47*/ { "Rd Intensity",FASTZ_OPCMD_RD_INTEN, {0},1, NULL, eOpModeCmd, FALSE },
/*  StgAF read inten */
      { NULL,             0,{0},  -1, NULL, eOpModeCmd,    FALSE },
} },
{ "PMT Amp",
{                          /* 8, PMT preamp controller       */
/* 0*/ { "SoftWr Ver",     0,{0},  1, NULL, eNetVarToSGI,  FALSE },  /*O SW Rev,     */
/* 1*/ { "ResetStatus", 0,{0},  1, NULL, eNetVarToSGI,  FALSE },       /*O ResetStat,   */
/* 2*/ { "ModeCmd",      0,{0},  1, NULL, eNetVarFromSGI,FALSE }, /*  Mode,  cmd  */
/* 3*/ { "Status",    0,{0},  1, NULL, eNetVarToSGI,  FALSE },          /*O Status,    */
/* 4*/ { "OpParArr1",      HDWR_DEF_PMT_AZFCT,     {0},   1, NULL,
eNetVarFromSGI,FALSE },        /*  cmd,      */
/* 5*/ { "OpParArr2",      0,{0},  1, NULL, eNetVarFromSGI,FALSE }, /*  cmd,       */
/* 6*/ { "OpStsArr1",      0,{0},  1, NULL, eNetVarToSGI,  FALSE },  /*O Status,    */
/* 7*/ { "OpStsArr2",      0,{0},  1, NULL, eNetVarToSGI,  FALSE },  /*O Status,    */
/* 8*/ { "ResetFocus",     HDWR_PMT_RST_FOCUS_OFF, {0}, 1, NULL,
eNetVarFromSGI,FALSE },        /*  resetFocus,      */
/* 9*/ { "ResetPeakDet", HDWR_PMT_RST_PKDET_OFF,  {0}, 1, NULL,
eNetVarFromSGI,FALSE },        /*  resetPeakDet,    */
/*10*/ { "FocThresStat", 0,{0},  1, NULL, eNetVarToSGI,  FALSE },       /*O
FocusThresholdExeceed */
/*11*/ { "PMTOverloaded",0,{0},  1, NULL, eNetVarToSGI,  FALSE },   /*O overload,  */
/*12*/ { "PMTClamped",    0,{0},  1, NULL, eNetVarToSGI,  FALSE },  /*O clamped,  */
```

```
/*13*/ { "AFThresCmd", HDWR_DEF_PMT_AFTHRESH, {0}, 1, NULL,
eNetVarFromSGI,FALSE }, /* set threshold */
/*14*/ { "PMTVoltCmd", HDWR_DEF_PMT_PMTVOLT, {0}, 1, NULL,
eNetVarFromSGI,FALSE }, /* set PMT voltage, */
/*15*/ { "AutoZeroCmd", HDWR_DEF_PMT_AZERO, {0}, 1, NULL,
eNetVarFromSGI,FALSE }, /* start autozero, */
/*16*/ { "AZeroGainK1", HDWR_DEF_PMT_AZEROGAINK1,{0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AutoZeroGainK1, */
/*17*/ { "AZeroGainK2", HDWR_DEF_PMT_AZEROGAINK2,{0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AutoZeroGainK2, */
/*18*/ { "AZeroGainK3", HDWR_DEF_PMT_AZEROGAINK3,{0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AutoZeroGainK3, */
/*19*/ { "AvgAutoGainK1",HDWR_DEF_PMT_AVGAGAINK1, {0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AvgAutoGainK1, */
/*20*/ { "AveAutoGainK2",HDWR_DEF_PMT_AVGAGAINK2, {0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AvgAutoGainK2, */
/*21*/ { "AveAutoGainK3",HDWR_DEF_PMT_AVGAGAINK3, {0}, 2, NULL,
eNetVarFromSGI,FALSE }, /*O AvgAutoGainK3, */
/*22*/ { "SelfTest", PMT_OPCMD_SELFTEST, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* SelfTest, cmd */
/*23*/ { "AutoZero", PMT_OPCMD_AUTOZERO, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* AutoZero, cmd */
/*24*/ { "PkAutoGain", PMT_OPCMD_PK_AGAIN, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* PeakAutoGain, cmd */
/*25*/ { "AvgAutoGain", PMT_OPCMD_AVG_AGAIN, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* AvgAutoGain, cmd */
/*26*/ { "ReadAvgVideo", PMT_OPCMD_RD_AVG_VIDEO, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* ReadAvgVideo, cmd */
/*27*/ { "ReadPkVideo", PMT_OPCMD_RD_PK_VIDEO, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* ReadPeakVideo, cmd */
/*28*/ { "ReadPMTHV", PMT_OPCMD_RD_PMT_HV, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* ReadPMTHV, cmd */
/*29*/ { "ReadLaserPwr", PMT_OPCMD_RD_LSR_PWR, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* ReadLaserPwr, cmd */
/*30*/ { "ReadVideoZero",PMT_OPCMD_RD_VIDEOZERO, {0}, 1, NULL, eOpModeCmd,
FALSE }, /* ReadVideoZero, cmd */
/*31*/ { "ReadFocus", PMT_OPCMD_RD_FOCUS, {0}, 1, NULL, eOpModeCmd,
FALSE },            /* ReadFocus,    cmd      */
/*32*/ { "ReadPmtOvld", PMT_OPCMD_RD_PMTOVLD,  {0}, 1, NULL, eOpModeCmd,
FALSE },            /* ReadPmtOvld, cmd     */
      { NULL,              0, {0},-1, NULL, FALSE, FALSE },
} },
{ "Turret/OpticsCtrl",
{                           /* 9, Turret/Optics controller              */
/* 0*/ { "SoftWr Ver",    0,{0}, 1, NULL, eNetVarToSGI,   FALSE },   /*O SW Rev,     */
/* 1*/ { "ResetStatus",0,{0}, 1, NULL, eNetVarToSGI,   FALSE },       /*O ResetStat,    */
/* 2*/ { "ModeCmd",      0,{0}, 1, NULL, eNetVarFromSGI, FALSE },  /*  Mode,  cmd   */
/* 3*/ { "Status",   0,{0}, 1, NULL, eNetVarToSGI,   FALSE },           /*O Status,    */
/* 4*/ { "PositionCmd",HDWR_TURRET_HOME_OBJ, {0}, 1, NULL, eNetVarFromSGI,
FALSE },            /*  TurretPosCommand           1-5        */
/* 5*/ { "WL/Laser Sw",HDWR_LSR_WHLT_WHITE,  {0}, 1, NULL, eNetVarFromSGI,
FALSE },            /*  ModeSw, White/Laser          1/0      */
/* 6*/ { "Shutter O/C",HDWR_LSR_SHUTTER_CLS, {0}, 1, NULL, eNetVarFromSGI, FALSE
},            /*  LaserShutter, open/close   1/0   */
/* 7*/ { "StepDelay",  HDWR_TURRET_STEPDELAY,{0}, 2, NULL, eNetVarFromSGI,
FALSE },            /*  TurretStepDelay  */
/* 8*/ { "ObjStatus",       0,{0}, 1, NULL, eNetVarToSGI,   FALSE },           /*O
ObjectiveSensorStat  on/off =???   */
/* 9*/ { "ClickStatus",1,{0}, 1, NULL, eNetVarToSGI,   FALSE },          /*O
clickSensorStatus      on/off= ???   */
/*10*/          { "HomeStatus", 1,{0}, 1, NULL, eNetVarToSGI,   FALSE },              /*O
HomeSensorStatus     on/off= ???   */
/*11*/          { "TurrPosStat",1,{0}, 1, NULL, eNetVarToSGI,   FALSE },              /*O
TurretPosStatus              1-5      */
      { NULL,             0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "Cassette Switch",
{                           /* 10, Cassette switch        */
/* 0*/ { "SoftWr Ver",    0,{0}, 1, NULL, eNetVarToSGI,   FALSE },   /*O SW Rev,     */
/* 1*/ { "ResetStatus",0,{0}, 1, NULL, eNetVarToSGI,   FALSE },       /*O ResetStat,    */
/* 2*/ { "ModeCmd",      0,{0}, 1, NULL, eNetVarFromSGI, FALSE },  /*  Mode,  cmd   */
/* 3*/ { "Status",   0,{0}, 1, NULL, eNetVarToSGI,   FALSE },           /*O Status,    */
/* 4*/ { "CassPresent",HDWR_CASS_NOT_PRES,{0}, 1, NULL, eNetVarToSGI, FALSE
},            /*O CassettePresent     Yes/no=???        */
```

```
/* 5*/ { "SizeStatus", HDWR_CASS_SIZE_8IN,{0}, 1, NULL, eNetVarToSGI,   FALSE },/*O
CassetteSizeStatus        0-3 */
/* 6*/  { "InterlockStat",0,{0},  1, NULL, eNetVarToSGI, FALSE },          /*  Mode, cmd   */
        { NULL,              0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "Solenoid Ctrl",
{                              /* 11, Solenoid controller   */
/* 0*/  { "SoftWr Ver",      0,{0},  1, NULL, eNetVarToSGI,   FALSE },  /*O SW Rev,     */
/* 1*/  { "ResetStatus", 0,{0},  1, NULL, eNetVarToSGI,   FALSE },       /*O ResetStat,    */
/* 2*/  { "ModeCmd",     0,{0},  1, NULL, eNetVarFromSGI, FALSE },    /*  Mode,  cmd  */
/* 3*/  { "Status",    0,{0},  1, NULL, eNetVarToSGI,   FALSE },         /*O Status,   */
/* 4*/ { "DoorCmd",         HDWR_SOL_WFDRCMD_CLS, {0}, 1, NULL, eNetVarFromSGI,
FALSE },           /*  CmdWaferDoor      Open/close= ??? */
/* 5*/ { "ChuckVacCmd",HDWR_SOL_CHKVAC_NONE, {0}, 1, NULL, eNetVarFromSGI,
FALSE },           /*  CmdWaferChuckSolenoid          0-3  */
/* 6*/ { "DoorStat",          HDWR_SOL_WFDRSTAT_CLS,{0}, 1, NULL, eNetVarToSGI,
FALSE },           /*O Stat WaferDoor     open/close=???   */
/* 7*/ { "ChuckVacStat",0,{0},  1, NULL, eNetVarToSGI,   FALSE },            /*O
StatChuckVac          on/off= ???         */
/* 8*/ { "HouseVacStat",0,{0},  1, NULL, eNetVarToSGI,   FALSE },            /*O
StatHouseVac          on/off= ???         */
        { NULL,              0,{0}, -1, NULL, FALSE, FALSE },
} },
{ "Aperture Ctrl",
{                              /* 11, Solenoid controller   */
/* 0*/  { "SoftWr Ver",      0,{0},  1, NULL, eNetVarToSGI,   FALSE },  /*O SW Rev,     */
/* 1*/  { "ResetStatus", 0,{0},  1, NULL, eNetVarToSGI,   FALSE },       /*O ResetStat,    */
/* 2*/  { "ModeCmd",     0,{0},  1, NULL, eNetVarFromSGI, FALSE },    /*  Mode,  cmd  */
/* 3*/  { "Status",    0,{0},  1, NULL, eNetVarToSGI,   FALSE },         /*O Status,   */
/* 4*/ { "Position Cmd",0,{0},  1, NULL, eNetVarFromSGI, FALSE },
/* 5*/ { "Pos status",  0,{0},  1, NULL, eNetVarToSGI,   FALSE },
/* 6*/ { "Home Status", 0,{0},  1, NULL, eNetVarToSGI,   FALSE },
/* 7*/ { "Step Count",  0,{0},  1, NULL, eNetVarToSGI,   FALSE },
/* 8*/ { "Travel Lim",  0,{0},  1, NULL, eNetVarFromSGI, FALSE },
        { NULL,              0,{0}, -1, NULL, FALSE, FALSE },
} },
{ NULL,
{
      { NULL,     0, -1, NULL, FALSE, FALSE },
} },
};



#define AF_OBJ_PWR_NEED_INTEN 10
/********************************************************************
 * stage autofocus
 * Laser Intensity/pmt gain setting are from configuration file
* Low power objectives: added extra travel
********************************************************************
 */
extern int lonui_StageAF(void)
  {
  UV_CONFIG       *cfg     = cfg_GetLockRdConfig();
  OPTICS_PARMS *pOptic        = da_GlobalLonPointer();
  int          OldLaserInten, AFLaserInten, OldPMTAmpAZero, iValue, iSpeed;
  BOOL              rt = FALSE;
  float        fZPos, fZLimUm, fCt2Um, fSpeed;
  int          iObj, iSpdMult, iOldYAmpl, iIntenOffset, iTimeCt, CurrFineZ, iObjPwr;

  lon_APIQuery( eLonAPITurretPosCmd,   &iObj,          0);
  iObj –;

  fZLimUm    = cfg->StageSpec.ZLimitUm[iObj] - pOptic->SampleThick;
  iObjPwr = cfg->ObjSpec.Power[iObj];

  CurrFineZ =  HDWR_FASTZ_MID_POS;
  lonui_SetFastZPos( CurrFineZ);

  if ( iObjPwr < AF_OBJ_PWR_NEED_INTEN &&
            cfg->StageSpec.StgAFLoMagAlgor == eStgAFLoMagMoveToLim )
    {
        fZPos = fZLimUm - 150.0;
        rt = IonuiMoveStageZUm( fZPos, 100, eStgSwitchAxis);
        }
      else
        {
        fCt2Um          = cfg->StageSpec.ZMStep2um;
        OldLaserInten = UIimage.LaserPwr;
        AFLaserInten = Ionui_GetIntensityFromLineObjSelected( pOptic, iObj, cfg);

        Ion_APIQuery( eLonAPIPageYAmpl,        &iOldYAmpl, 0);
        Ion_APICmd  ( eLonAPIPageYAmpl,        5);

        OldPMTAmpAZero = -1;
        if( Ion_APIQuery( eLonAPIPMTAmpAZeroCmd, &iValue, 0) == TRUE)
          OldPMTAmpAZero = iValue ;

        IonuiAZFunction( cfg->StageSpec.StgAF_Zero);
        Ionui_LaserPwrAndPMT( AFLaserInten);

       if( iObjPwr < AF_OBJ_PWR_NEED_INTEN &&
                        cfg->StageSpec.StgAFLoMagAlgor == eStgAFLoMagIntArray)
          IonuiLoPwrStgAF( fCt2Um, fZLimUm, AFLaserInten, cfg->ObjSpec.NA[iObj]);
        else
          {
          rt = stg_ReadStageZUm( &fZPos, TRUE);
          if( rt == TRUE && (int)fZPos > BACKUP_BEFORE_AF_UM )
              fZPos = IonuiMoveStgDownBeforeAF( fZPos, fZLimUm, iObjPwr,
eStgUseCurrentAxis);

          iSpeed = (int)( (double)fZLimUm - fZPos);
          if( iSpeed < 1000)
            mot_SetCurrAxisSpeed( (int)( (double)iSpeed / fCt2Um));

          if( rt == TRUE)
            rt = Ionimg_OneAFCycle(  fZLimUm);
fprintf( stderr, "1rt=%d,", rt );
          if( rt == TRUE)
```

```
            rr = lonuiStgAF2And3( OldLaserInten, AFLaserInten, fZLimUm, fCt2Um, iSpdMult,
  iObjPwr );
          }
      lonui_LaserPwrAndPMT( OldLaserInten);
      if( OldPMTAmpAZero > 0)
        lon_APICmd( eLonAPIPMTAmpAZeroCmd, OldPMTAmpAZero);

      mot_SetDefaultSpeed( eAxisZ, FALSE);
      if( rr == TRUE && iObjPwr >= OBJ_PWR_FAF_AFTER_STGAF )
         CurrFineZ = lonuiLocalFineAF();
      lon_APICmd  ( eLonAPIPageYAmpl, iOldYAmpl);
      }
    mot_EnableCurrentAxisJoystick();
    return CurrFineZ;
    }

 /***********************************************
  * do stage AF at low power obj
  * steps:
  *      1: go to fix position, LO_MAG_AF_RANGE below limit
  *      2, Move to limit while watching for intensity
  *      3, based on intensity read, set new laser intensity and
  *             calc first gauss target
  *      4, start from LO_MAG_2NDPASS_RANGE above and go to
  *                  LO_MAG_2NDPASS_RANGE below, while watching
  *             for intensity.
  *      5, based on intensity read, calc and goto final target
  *    4a, if intensity read from 1st search pass is all 0
  *        the second search range is LO_MAG_LO_LITE_RANGE
  ***********************************************
  */
 #define LO_MAG_AF_RANGE     1000.0
 #define LO_MAG_LO_LITE_RANGE     400.0
 #define LO_MAG_2NDPASS_RANGE     120.0
 #define LO_MAG_2NDPASS_2RANGE   ( 2 * LO_MAG_2NDPASS_RANGE)

 static BOOL lonuiLoPwrStgAF( float fCt2Um, float fZLimUm, int LastLaserI, float fNA)
```

```
{
float       fZPos, fMedPosUm, fTarget, fDelta;
int         iMaxInten, iRange, iIPeak;
double      dHalfRange2ndPass;
BOOL        rt, bPMTClamped = FALSE;

dHalfRange2ndPass = LO_MAG_2NDPASS_RANGE;
if( fNA > 0.01 && fNA < 0.15)
  {
  dHalfRange2ndPass = 2.0 / ( (double)fNA * fNA);
  if( dHalfRange2ndPass > (LO_MAG_AF_RANGE /2) )
     dHalfRange2ndPass = (LO_MAG_AF_RANGE / 2);
  }
fZPos = fZLimUm - LO_MAG_AF_RANGE;
rt = IonuiMoveStageZUm( fZPos, 100, eStgSwitchAxis);
if( rt == TRUE)
  rt = mot_SetCurrAxisSpeed( (int)( LO_MAG_AF_RANGE/ fCt2Um));
if( rt == TRUE)
  rt = IonuiMoveStgAndReadInten( fZPos, fZLimUm, &iMaxInten, &fMedPosUm, FALSE);
if( rt == TRUE)
  {
  Ion_APIQuery( eLonAPIPMTAmpClamped, &bPMTClamped, 0);
  if( bPMTClamped == TRUE)
     iMaxInten = 250;

  iIPeak = 2 * iMaxInten ;
  if( iIPeak > 400)
     iIPeak = 400;
  if( iIPeak < 20)
     iIPeak += 20;
  else
  if( iIPeak < 50)
     iIPeak += ( 20 * (50 - iIPeak)/30);

  zr_SetTargetLaserIFrImgInten( iIPeak, LastLaserI);
  fTarget = (double)fZPos + fMedPosUm;
  rt = stg_ReadStageZUm( &fZPos, FALSE);
```

```
        if( iMaxInten < 1 || bPMTClamped == TRUE)
          fTarget = fZLimUm - LO_MAG_LO_LITE_RANGE;
        }
      if( rt == TRUE && fZPos > (double)fTarget)
        {
        if( iMaxInten > 0 && bPMTClamped != TRUE)
          {
          fDelta = (double)fZPos - fTarget;
          if( fDelta > dHalfRange2ndPass)
            {
            fZPos = fZPos+ dHalfRange2ndPass- fDelta;
            rt = IonuiMoveStageZUm( fZPos, 100, eStgUseCurrentAxis);
            }
          fTarget = fZPos - ( dHalfRange2ndPass + dHalfRange2ndPass);
          }
        if(rt == TRUE)
          {
          iRange = (int)fZPos - (int)fTarget;
          mot_SetCurrAxisSpeed( (int)( (double)iRange / fCt2Um));
  fprintf( stderr, "fr=%d,", (int)fZPos);
          rt = IonuiMoveStgAndReadInten( fZPos, fTarget, &iMaxInten, &fMedPosUm, FALSE);
          }
        if( rt == TRUE)
          {
          fTarget = (double)fZPos - fMedPosUm;
          rt = IonuiMoveStageZUm( fTarget, 100, eStgUseCurrentAxis);
  fprintf( stderr, "Target=%d\n", (int)fTarget);
          }
        }
    return rt;
    }

  /**************************************************
  * move stage down: ensure start from below focus
  **************************************************
  */
```

```
static float lonuiMoveStgDownBeforeAF( float fZPos, float fZLimUm, int iObjPwr,
                                        STG_ISSUE_SW_AXIS_CMD AxisCmd)
  {
  int   iTimeCt;

  if( (int)fZPos > (int)fZLimUm)
     fZPos = fZLimUm;
  if( iObjPwr <= AF_OBJ_PWR_NEED_INTEN )
     fZPos -= WORKING_DIST_10X_UM;
  else
     fZPos -= WORKING_DIST_UM;
  lonuiMoveStageZUm( fZPos, 100, AxisCmd);
  return fZPos;
  }


/************************************************************
 * do step 2 and 3 of Stage AF
 ************************************************************
 */
static BOOL lonuiStgAF2And3( int OldLaserInten, int AFLaserInten, float fZLimUm, float
                                        fCt2Um, int iSpdMult, int iObjPwr )
  {
  BOOL          rt = FALSE;
  float         fZPos, fCurrLim;
  int           iOffset, iSpeed, iDiffUm;
  UV_CONFIG     *pCfg    = cfg_GetLockRdConfig();

  rt = stg_ReadStageZUm( &fZPos, FALSE);
  if( rt == TRUE)
    {
    if( fabs( (double)fZPos - fZLimUm) > 50)
       iOffset = -1200;
    else
       iOffset = -200;
    }
  if( rt == TRUE)
```

```
        {
        rt = lonuiAutoFocusCycle23( &AFLaserInten, iOffset, fZLimUm, fCt2Um, iSpdMult,
   fZPos);
   fprintf( stderr, "2rt=%d,", rt);
        if( rt == TRUE)
           rt = stg_ReadStageZUm( &fZPos, FALSE);
        if( rt == TRUE)
           rt = lonuiAutoFocusCycle23( &AFLaserInten,  140, fZLimUm, fCt2Um, iSpdMult,
  fZPos);
        }
  fprintf( stderr, "3rt=%d,", rt);
    return rt;
    }

 /*
       if ( iObjPwr < AF_OBJ_PWR_NEED_INTEN &&
                 pCfg->StageSpec.StgAFLoMagAlgor == eStgAFLoMagIntArray)
         {
 fprintf( stderr, "Pos=%d,Inten=%d,", (int)fZPos, AFLaserInten );
         iOffset = -200;
         fCurrLim = (double)fZPos + iOffset;
         if( fCurrLim < 0.0)
           fCurrLim = 0;
         if( fCurrLim > (double)fZLimUm)
           fCurrLim = fZLimUm ;
        iDiffUm = abs( (int)( (double)fCurrLim - fZPos) );
        iSpeed  =  iDiffUm;
        if( iSpeed < 50)
          iSpeed = 50;
       rt = lonuiAFSetLaserIntenAndSpeed( &AFLaserInten, (int)( (double)iSpeed / fCt2Um) );
       if( rt == TRUE)
         rt = lonuiLoObjPwrStgAF2And3(OldLaserInten, fZPos, fCurrLim, FALSE);
       }
*/

/************************************************************
* run the 2nd, 3rd cycle of the autofocus
```

```
 ****************************************************************
 */
static BOOL IonuiAutoFocusCycle23( int *AFLaserInten, int Offset, float fZLim, float fCt2Um,
                                    int iSpdMult, float fZPos )
{
BOOL rt = FALSE;
float       fZCurrLim;
int         iDiffUm, iSpeed;

  {
  fZCurrLim = (double)fZPos +  Offset;
  if( fZCurrLim < 0.0)
     fZCurrLim = 0;
  if( fZCurrLim > (double)fZLim)
     fZCurrLim = fZLim;
  iDiffUm = abs( (int)( (double)fZCurrLim - fZPos) );
  iSpeed  = iDiffUm / 2;                              /* 2second to travell */
  if( iSpeed < 25)
     iSpeed = 25;
  rt = IonuiAFSetLaserIntenAndSpeed( AFLaserInten, (int)( (double)iSpeed / fCt2Um) );
fprintf( stderr, "Pos=%d,Inten=%d,", (int)fZPos, *AFLaserInten );
  }
if( rt == TRUE)
  rt = Ionimg_OneAFCycle( fZCurrLim);
return rt;
}

/**************************************************************
 * stage AF step 2 and 3 for low power obj
 * Step 2: move Z to target position, and at the same time read
 *  intensity. At the end of the movement, get the readings
 *  and determine, based on intensity array, where the peak is
 *  Peak is determined to be at the median point
 * Step 3: base on the step 2 intensity array, in-focus
 *  stage position is determined, and stage is moved there.
 *  Before moving, laser intensity, stage speed are restored.
 *  Restore laser intensity to prevent PMT overload, restore
 * speed to improve throughput.
 *
 ************************************************************
 */
static BOOL IonuiLoObjPwrStgAF2And3( int OldLaserInten, float fCurrZUm, float fZLimUm,
                                              BOOL        bWakeAxis)
  {
  BOOL      rt, bMotoring;
  int       iMaxInten, iCt;
  float     fMedPosUm, fNewTarget;

  rt = IonuiMoveStgAndReadInten( fCurrZUm, fZLimUm, &iMaxInten, &fMedPosUm,
bWakeAxis);
  Ionui_LaserPwrAndPMT( OldLaserInten);
  if( rt == TRUE )
    {
    fNewTarget = (double)fCurrZUm - fMedPosUm;
    if( fNewTarget < 0.0)
      fNewTarget = 0;
fprintf( stderr, "target=%d:%d, ", (int)fMedPosUm, (int)fNewTarget );
    mot_SetDefaultSpeed( eAxisZ, FALSE);
    rt = stg_MoveStageZUm( fNewTarget, FALSE);
    for( iCt= 0, bMotoring = TRUE; iCt< 20 && bMotoring == TRUE; iCt++)
      {
      bMotoring = mot_IsMotorMoving();
      if( bMotoring == TRUE)
        util_sginap( 2);
      }
    if( bMotoring == TRUE)
      {
      mot_StopCurrentAxis();
      rt = FALSE;
      }
    }
 return rt;
 }
```

```
typedef unsigned int UINT;
#define STG_AF23_ARRAYS 5
#define AF23_SCALE       80      /* suppose to read 80, z stops at more/less 80 */
/***********************************************
 * move stage and watch for intensity
 * check for cases where sum = 1, or only one nonzero
 *           number
 ***********************************************
 */
static BOOL lonuiMoveStgAndReadInten( float fCurrZUm, float fZLimUm, int *iMaxInten,
                                      float *fMedPosUm, BOOL bWakeAxis )
 {
 BOOL          rt = FALSE, bMotoring;
 int       iValue, iCt, iArrayCt;
 BYTE          chArray  [STG_AF23_ARRAYS][LON_ARRAY_DATA_SZ];
 int       *iArray;
 int       *iFiltered;

 int       iSum, iMedian, iMedCt, iNumArrays, iMaxI, iArraySize;
 float     fZScale, fOffset, fNewTarget;

 iArraySize = STG_AF23_ARRAYS * LON_ARRAY_DATA_SZ;
 iArray    = (int *)XtMalloc( sizeof(int) * iArraySize);
 iFiltered = (int *)XtMalloc( sizeof(int) * iArraySize);

 iNumArrays = STG_AF23_ARRAYS;
 longmi_UpdateNV( FAST_Z_NODE_ADDR, FASTZ_OPCMD_ARR2_NV_ADDR, 6);
 longmi_UpdateNV( FAST_Z_NODE_ADDR, FASTZ_OPCMD_ARR1_NV_ADDR,
LON_ARRAY_DATA_SZ * iNumArrays);
 fOffset  = 5;
 fZScale  = ( (double)fCurrZUm - fZLimUm) / AF23_SCALE;
 fZScale  = fabs( fZScale);
fprintf( stderr, "Target=%d,", (int)fZLimUm );
 rt = stg_MoveStageZUm(fZLimUm, bWakeAxis);
 rt = lonimg_QueryNetVar( FAST_Z_NODE_ADDR, FASTZ_MODE_RD_INTENSITY,
&iValue, TRUE);
```

```
  bMotoring = mot_IsMotorMoving();
  if( bMotoring == TRUE)
    mot_StopCurrentAxis();
fprintf( stderr. "motoring=%d,". bMotoring );
  iSum = 0;
  iMaxI = 0;
  *iMaxInten = iMaxI;

  for( iCt = 0; iCt < iNumArrays && rr == TRUE; iCt++)
    {
    if( lonimg_QueryNetVar16( FAST_Z_NODE_ADDR. FASTZ_INTEN_ARRAY0 + iCt,
                          &iValue. (char *)chArray[ iCt]. TRUE) !=
LON_ARRAY_DATA_SZ)
      rr = FALSE;
    for( iArrayCt= 0; iArrayCt < LON_ARRAY_DATA_SZ; iArrayCt++)
      {
      if( iMaxI < (UINT)chArray[iCt][iArrayCt])
        iMaxI = chArray[iCt][iArrayCt];
      iArray[ iCt * LON_ARRAY_DATA_SZ + iArrayCt] = chArray[iCt][iArrayCt];
      }
    }
 if( rr == TRUE)
    {
    *iMaxInten = iMaxI;
    if( iMaxI < 3)
      memcpy( iFiltered. iArray. sizeof(int) * iArraySize);
    else
      {
      iFiltered[0] = (3 * iArray[0] + iArray[1]) /4;
      for( iCt = 1; iCt < (iArraySize -1); iCt++)
        iFiltered[iCt] = (iArray[iCt -1] + (2 * iArray[iCt]) + iArray[iCt +1]) /4;
      iFiltered[iArraySize-1] = (3 * iArray[iArraySize -1] + iArray[iArraySize -2] )/4;

      iMaxI = 0;
      for( iCt = 0; iCt < iArraySize; iCt++)
        {
        if( iMaxI < iFiltered[iCt])
```

```
                iMaxI = iFiltered[iCt];
              }
            *iMaxInten = iMaxI;
            }
          for( iCt =0; iCt < iArraySize; iCt++)
            iSum += iFiltered[ iCt];

          if( iSum > 1)
            iSum = iSum /2;

          iMedian = 0;
          for( iCt = 0; iCt < iArraySize && iMedian < iSum; )
            {
            iMedian +=  iFiltered[ iCt];
            if( iMedian < iSum)
              iCt ++;
            }
          *fMedPosUm = fOffset + ( (double)fZScale * iCt);
          }

      fprintf( stderr, "\nPos=%d,", iCt);
      for( iCt = 0; iCt < iArraySize; iCt++)
        {
        fprintf( stderr, "%d ", iFiltered[ iCt]);
        if( (iCt %16) == 15)
          fprintf( stderr, "\n");
        }
      fprintf( stderr, "\nMaxI = %d,", iMaxI);

        XtFree( (char *)iArray);
        XtFree( (char *)iFiltered);
        return ( rt);
        }


      /*************************************************************
```

```
 * support functions for Autofocus
 * set laser intensity, Z speed
 * after each trial intensity is reduced
 ************************************************************
 */
static BOOL lonuiAFSetLaserIntenAndSpeed( int *AFLaserInten, int SpeedCt)
  {
  BOOL          rt;
  int           iValue;

  rt = lon_APIQuery( eLonAPIPMTAmpClamped, &iValue, 0);
  if( rt == TRUE &&  iValue == PMT_CLAMPED)
    *AFLaserInten = *AFLaserInten * AF2_INTEN_DEC;
  else
    *AFLaserInten = *AFLaserInten * AF2_INTEN_DEC_NORMAL;
  rt =lonui_LaserPwrAndPMT( *AFLaserInten);

  if( rt == TRUE)
    mot_SetCurrAxisSpeed( SpeedCt);
  return rt;
  }
/************************************************************
 * doing an autozero function is very slow, so just set it
 ************************************************************
 */
static BOOL lonuiAZFunction( int  iAZero)
  {
  lon_APICmd( eLonAPIPMTAmpAZeroCmd,   iAZero);

  return TRUE;
  }

/**********************************************
 * move Zand wait till done
 **********************************************
 */
```

```
static BOOL lonuiMoveStageZUm( float fZPos. int iMaxCt. STG_ISSUE_SW_AXIS_CMD
AxisCmd)
  {
  BOOL      rt ;
  int       iTimeCt;

  rt = stg_MoveStageZUm( fZPos. (AxisCmd == eStgUseCurrentAxis)? FALSE: TRUE );
  if( rt == TRUE)
    {
    iTimeCt = 0;
    while( mot_IsMotorMoving() == TRUE && iTimeCt < iMaxCt)
      {
      iTimeCt++;
      util_sginap(3);
      }
    if( iTimeCt >= iMaxCt)
      {
      rt = FALSE;
      mot_StopCurrentAxis();
      fprintf( stderr, "time=%d,", iTimeCt);
      }
    }
  return rt;
  }

/**********************************************************
 * one cycle auto focus
 * 1, reset AF flipflop,
 * 2, move Z,
 * 3, wait till Ztarget done or AF threshold exceeded
 **********************************************************
 */
extern BOOL lonimg_OneAFCycle( float  fZLimUm)
  {
  BOOL              rt = FALSE, bMotoring;
  HARDWR_IMAGE_ENTRY   *FastZImg, *PMTImg;
  int         iValue, iTimeCt;
  FastZImg = &( HardWrImgTable[ FAST_Z_NODE_ADDR].NVImg[
FASTZ_OPSTATUS_NV_ADDR]);
  PMTImg   = &( HardWrImgTable[ PMT_AM_NODE_ADDR].NVImg[
PMTAM_CLAMPED_NV_ADDR]);
  rt = lonimg_QueryNetVar( FAST_Z_NODE_ADDR,
FASTZ_MODE_WAITFOCUS_NV_ADDR, &iValue, TRUE);
if( rt == FALSE)
fprintf( stderr, "cannot send WaitForFoc,");
else
if( iValue != C_AF_COMPARATOR_RDY)
fprintf( stderr, "WaitForFoc=%d,", iValue);

  if( rt == TRUE && iValue == C_AF_COMPARATOR_RDY)
    {
fprintf( stderr, "Tar=%d,", (int)fZLimUm );

    FastZImg->UpdatedFlg = FALSE;
    PMTImg->UpdatedFlg = FALSE;
    rt = stg_MoveStageZUm(fZLimUm, FALSE);
if( rt == FALSE)
fprintf( stderr, "StageMoveErr,");
    if( rt == TRUE)
      {
      iTimeCt = 0;
      for( bMotoring= TRUE; bMotoring == TRUE && FastZImg->UpdatedFlg == FALSE
                 && PMTImg->UpdatedFlg == FALSE && iTimeCt < 100; )
        {
        util_sginap(1);              /* dont put this at the end of the loop, too slow */
        LonTimeBlockReadOneAndDecodeMsg(AF_READ_TICK);
        if( PMTImg->UpdatedFlg == FALSE &&  FastZImg->UpdatedFlg == FALSE)
          bMotoring = mot_IsMotorMoving();
fprintf( stderr, "move=%d,", bMotoring );
        iTimeCt++;
        }
      if( bMotoring == TRUE)
        mot_StopCurrentAxis();
        if( PMTImg->UpdatedFlg == FALSE &&
            ( FastZImg->UpdatedFlg == FALSE || FastZImg->iNVValue !=
C_AF_COMP_TRIP) )
            rt = FALSE;

    if( PMTImg->UpdatedFlg == TRUE )
    fprintf( stderr, "PMT-OL=%d,", PMTImg->iNVValue);

    if( FastZImg->UpdatedFlg == FALSE)
    fprintf( stderr, "NoResposeFrFastZ,");
    else
    if( FastZImg->iNVValue != C_AF_COMP_TRIP)
    fprintf( stderr, "FastZResp=%d,", FastZImg->iNVValue );

    fprintf( stderr, "Time=%d,", iTimeCt );
        }
      if( FastZImg->UpdatedFlg == FALSE || FastZImg->iNVValue != C_AF_COMP_TRIP )

        lon_APICmd( eLonAPIFastZAbort, 0);
      }
  return rt;
  }
/**************************************************
 * calc new laser intensity based on image intensity
 * feedback
 * image Intensity here is heavily filtered, and is
 * usually lower than actual peak
 * if saturated, cut laser intensity by 5
 * if less than ideal, incr of 10( maybe in config) laser Inten = 100% img Inten
 * for incr of less the 10 laser inten, each step = 10%( linear, not expotenial)
 * return TRUE if
 **************************************************
 */
#define IDEAL_IMG_INTEN          195
#define HALF_OF_IDEAL        (IDEAL_IMG_INTEN /2)
#define MAX_TARGET_LASER_INTEN  75
```

```
extern BOOL zr_SetTargetLaserIFrImgInten( int iImgI, int CurrLaserI)
  {
  UV_CONFIG     *pCfg   = cfg_GetLockRdConfig();
  float         fNew, fCt;
  BOOL          rt = FALSE;
  int           iNewI;

  if( iImgI > IDEAL_IMG_INTEN)
    fNew = CurrLaserI - (iImgI - IDEAL_IMG_INTEN) /10;
  else
    {
    fNew = CurrLaserI;
    if( iImgI < 15)
      iImgI = 15;
    for( ; iImgI < HALF_OF_IDEAL; iImgI += iImgI)
      fNew += (double)pCfg->SysSpec.ZRangeLaserGain;
    fCt = (double)pCfg->SysSpec.ZRangeLaserGain * (IDEAL_IMG_INTEN - iImgI) / iImgI;

    fNew += (double)fCt;
    if( (int)fNew > MAX_TARGET_LASER_INTEN)
      fNew = MAX_TARGET_LASER_INTEN;
    rt = TRUE;
    }
  iNewI = fNew;
  if(iNewI < 0)
    iNewI = 0;
  lomui_LaserPwrAndPMT( iNewI);
fprintf( stderr, "newLI=%d, ", iNewI);
  return TRUE;
  }
/*****************************************************************
 * get laser intensity seting  based on current laser line selected
 *****************************************************************
 */
static int lomui_GetIntensityFromLineObjSelected( OPTICS_PARMS *pOptic,
                                     int iObj, UV_CONFIG *pCfg)
  {
    int         iIntenOffset, *pIOffset, iInten;

    switch( UIImage.eLaserLine)
      {
      case  eSrc458:  iInten = pOptic->AFInten458; break;
      case  eSrc488:  iInten = pOptic->AFInten488; break;
      case  eSrc515:  iInten = pOptic->AFInten515; break;
      case  eSrcWhite:  iInten = pOptic->AFIntenAll; break;

      default:        iInten = pOptic->AFInten488;
                      break;
      }
    pIOffset = Ionui_GetLaserIOffsetForLaserLine( &pCfg->ObjSpec, UIImage.eLaserLine);
    iIntenOffset = pIOffset[iObj] - pIOffset[0];
    iInten = iInten + iIntenOffset;
    if( iInten < 0)
      iInten = 0;
    if( iInten > 100)
      iInten = 100;
    return iInten;
    }
/******************************************************************
 * get the Laser I offset array based on laserLine specified
 ******************************************************************
 */
extern int * Ionui_GetLaserIOffsetForLaserLine( UV_OBJ_SPEC *ObjSpec,
OPTICS_LASER_SRC LineFilter)
  {
  int * pOffset;

  switch ( LineFilter)
    {
    case eSrc458:
      pOffset = ObjSpec->LaserI458Offset;
      break;
    case eSrc488:
      pOffset = ObjSpec->LaserI488Offset;
```

```
        break ;
      case eSrc515:
        pOffset = ObjSpec->LaserI515Offset;
         break;
      case eSrcWhite:
      default:
        pOffset = ObjSpec->LaserIAllOffset;
        break ;
        }
   return pOffset;
   }
/*********************************************************************
 * change the laser intensity ( PMT gain and laser power) only,
 * no laser line change
 *
 * Specified LaserInten = 0-100
 * laser Power is set according to config, based on current laserline used
 *
 * if( LaserInten > LASER_I_PERCENT_CHG_PWR, laser power is increased upto
MAX_LASER_PWR
 *********************************************************************
 */
extern BOOL lonui_LaserPwrAndPMT( int LaserInten)
  {
  BOOL        rt = FALSE;
  int         iValue, LaserPwrRaw, PMT, MinPMTGain, PMTRange;
  UV_CONFIG  *cfg    = cfg_GetLockRdConfig();
  OPTICS_LASER_SRC LaserLine;


  if( LaserInten > 100)
    LaserInten = 100;
  if( LaserInten < 0)
    LaserInten = 0;

  lon_APIQuery( eLonAPILaserLineFilterMove, (int *)&LaserLine, 0);
  switch( LaserLine)
```

```
          {
          case  eSrc458:  LaserPwrRaw = cfg->ConfocalSpec.LaserPwr458;
                                break;
          case  eSrc488:  LaserPwrRaw = cfg->ConfocalSpec.LaserPwr488;
                                break;
          case  eSrc515:  LaserPwrRaw = cfg->ConfocalSpec.LaserPwr515;
                                break;
          case  eSrcWhite:
          default:          LaserPwrRaw = cfg->ConfocalSpec.LaserPwrAll;
                                break;
          }
       if( LaserInten > LASER_I_PERCENT_CHG_PWR && LaserPwrRaw < MAX_LASER_PWR
    )
          {
         LaserPwrRaw += (double)( LaserInten - LASER_I_PERCENT_CHG_PWR) *
                        ( MAX_LASER_PWR - LaserPwrRaw) / ( 100 -
    LASER_I_PERCENT_CHG_PWR );
          }
       rt = lon_APIQuery( eLonAPILaserOnOff, &iValue, 0);
       if( rt == TRUE  && iValue == HDWR_LASER_OFF)
         rt = lon_APICmd( eLonAPILaserOnOff, HDWR_LASER_ON);
       if( rt == TRUE)
         rt = lon_APIQuery( eLonAPILaserRunIdle, &iValue, 0);
       if( rt == TRUE  && iValue == HDWR_LASER_IDLE)
         rt = lon_APICmd( eLonAPILaserRunIdle, HDWR_LASER_RUN);
       if( rt == TRUE)
         {
         MinPMTGain = MIN_PMT_GAIN + cfg->ConfocalSpec.PMTMinGainOffset;
         PMT = MinPMTGain - ((LaserInten * cfg->ConfocalSpec.PMT_Range)/
    LASER_PWR_SLIDER_RANGE);

         if( PMT < 50)        /* safety */
            PMT = 50;
         if( PMT > 255)
            PMT = 255;
         if( LaserPwrRaw > MAX_LASER_PWR)
            LaserPwrRaw = MAX_LASER_PWR;
```

```
        if( LaserPwrRaw < 0)
           LaserPwrRaw = 0;
        rt = lon_APICmd( eLonAPILaserPwr, LaserPwrRaw);
        if( rt == TRUE)
           rt = lon_APICmd( eLonAPIPMTAmpVoltCmd, PMT);

        UIImage.LaserPwr      = LaserInten;
       }
     return rt;
    }
  ☐
```

```
/**************************************************
 * Ultrapointe Corp. copyright 1992,93,94
 * stage.c
 *
 * UltraView1000 software
 * this module handles the high level stage interface:
 * stage coordinates are in microns, with XYoffset from different Objective
 * reference is based on 2nd obj: Obj1, ie: no correction is zero for obj1
 * If currently using other objective, the XOffsetCt/YOffsetCt from .ucvonfig
 * is used to compensate
 *
 **************************************************
 */
#ifdef Irix5plus
#define NeedFunctionPrototypes 1
#else
#define FUNCPROTO
#endif

#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/signal.h>
#include <math.h>
#include <unistd.h>
#include <limits.h>                /* sginap */
#include <X11/Intrinsic.h>
#include "uv_util.h"
#include "uvconfig.h"
#include "wfrfile.h"
#include "uv_data.h"
#include "motor.h"
#include "stage.h"
#include "deskew.h"
#include "stgchild.h"
#include "lon_msg.h"
```

```
#include "lon_img.h"
#include "lonuiimg.h"
#include "lon_gmi.h"

#define RD_DELAY_TIME             60      /* in ms */
#define RD_DELAY_TICK             (RD_DELAY_TIME* CLK_TCK / 1000)
#define HOME_XY_TIMEOUT_CT        10

#define MAX_NO_MOVE_UM             1.0
#define MAX_NO_MOVE_ECT           2
#define NUM_CLOSED_LOOPS    6
#define MAX_ERROR_LIM_UM   2000.0

#define OBJ_MOVE_CLOSE_ROBOT     0      /* first object as reference */
#define MIN_XY_SPEED_UM          100

#define SENDER 1          /* direction for pipe */
#define RECEIVER 0
#define FILE_PTR int               /* file table index */


#define BACKLASH_UM      40  /* amount of xy backlash    */
                            /* see  EndPtPosCorrection()       */
#define INIT_BACKLASH_UP_UM 3000  /* back-and-fore stage init         */
#define INIT_BACKLASH_DN_UM 1000  /* back-and-fore stage init         */

#define MAX_REL_MOVE   25000      /* max rel move is 20000um each time */
#define STG_POS_END_POINT    120000.0  /* ends at +/- 120mm */
#define STG_NEG_END_POINT   -120000.0
#define STG_XY_TRAVEL_LIM    110000              /* travel limit, with offset */
#define STG_SQ_XY_TRAVEL_LIM     (110000.0 * 110000.0)

#define STG_Z_TRAVEL_LIM             12000                /* um */

static STAGE_STATUS StageInfo;
static BOOL stgBackgroundProc(STAGE_STATUS *pStgInfo, int iObj);
static BOOL stgEndPtPosCorrection( MOT_AXIS Axis, double TargetUM, int iObj,
                                        float EResol, float MResol);
static BOOL stg_MoveStageTillDone( STAGE_XY *StageXY, STG_MOVE_MODE MoveMode,
int iObj,
                                        int XMOffset, int YMOffset);
static int    stgUM2MotorCount( MOT_AXIS Axis, double um, int iObj);
static double stgEncoderCount2UM( MOT_AXIS Axis, double dCounts, int iObj);
static int    stgUM2EncoderCount( MOT_AXIS Axis, double um, int iObj);
static int stgObjMOffsetCt( MOT_AXIS Axis, UV_STAGE_SPEC *StageSpec, int iObj);
static int stgObjEOffsetCt( MOT_AXIS Axis, UV_STAGE_SPEC *StageSpec, int iObj);
static BOOL stg_RdStgXYPosAndMotorEncoderOffset( STAGE_XY *StageXY,
                                STG_MOVE_MODE MoveMode,
                                            int iObj,
                                            int *XMotorCtOffset, int *YMotorCtOffset);
static BOOL  stgSafeMoveXYStageRawAndNormal( STAGE_XY *StageXY, int iObj,
                                STG_MOVE_F_MODE eFMode, STG_MOVE_MODE MoveMode,
                                STG_MOVE_JOY_STATUS eMoveJoy );
static BOOL stgOrthoCorrAndDskToAppl( STAGE_XY *DestXY, STAGE_XY *SrcXY,
STG_MOVE_MODE MoveMode);
static BOOL stgOrthoCorrAndDskToStg ( STAGE_XY *DestXY, STAGE_XY *SrcXY,
STG_MOVE_MODE MoveMode);
static double stgIdealUMToMappedUM( MOT_AXIS Axis, UV_CONFIG *pCfg, double
                                dIdealUM);

static BOOL DebugPrtCh( char ch);


/****************************************************
 * stg_Initialize( void)
 * initialize XYZ stage
 ****************************************************
 */
extern BOOL stg_Initialize( int hUldd)
  {
  UV_STAGE_SPEC StageSpec;

  StageInfo.StgStatus    = eStgIdle;
  StageInfo.pidStage = -1;
```

```
    cfg_CopyStageConfig( &StageSpec);
    mot_Initialize( hUldd);
    dsk_ResetDeskewMatrix();
    StageInfo.CurrentXY.x  = 0;
    StageInfo.CurrentXY.y  = 0;
    StageInfo.TargetXY.x   = 0;
    StageInfo.TargetXY.x   = 0;
    StageInfo.BacklashXY.x = 0;
    StageInfo.BacklashXY.x = 0;
    StageInfo.CurrentZ          = 0;
    StageInfo.TargetZ      = 0;
    StageInfo.BacklashZ         = 0;
    StageInfo.bStage     = FALSE;
    StageInfo.XMResol           = StageSpec.XMStep2um;
    StageInfo.YMResol           = StageSpec.YMStep2um;
    StageInfo.ZMResol           = StageSpec.ZMStep2um;
    StageInfo.XEResol           = StageSpec.XEStep2um;
    StageInfo.YEResol           = StageSpec.YEStep2um;
    StageInfo.ZEResol           = StageSpec.ZEStep2um;
    StageInfo.BacklashCts  = (double)BACKLASH_UM /StageSpec.XMStep2um;
    return TRUE;
    }

/****************************************************
 * cleanup before program ends
 ****************************************************
 */
extern BOOL stg_TerminateStage( void)
   {
   mot_ClosePort();
   return TRUE;
   }

extern BOOL stg_IsStageOnline( void)
   {
   BOOL rt = FALSE;
   rt = StageInfo.bStage;
```

```
      return (rt);
      }

    /**********************************************************************
     * return TRUE if stage is put online
     * turret is also initialized here: home turret and then turn to low mag
     * set global update message
     *
     **********************************************************************
     */
    extern BOOL stg_PutStageOnline( void)
      {
      int Pos, j, iJGain;
      UV_CONFIG       *pCfg    = cfg_GetLockRdConfig();
      BOOL       rt;
      BYTE ch;

      if( StageInfo.StgStatus  == eStgIdle )
        {
        StageInfo.bStage = mot_PutStageOnline();

        rt = StageInfo.bStage;
        if( rt == TRUE)
          rt = mot_MoveToHome( eAxisZ);
        longmi_UpdateNV( TURRET_NODE_ADDR, TURRET_POS_CMD_NV_ADDR,
    HDWR_TURRET_HOME_OBJ);
        longmi_UpdateNV( TURRET_NODE_ADDR, TURRET_POS_CMD_NV_ADDR,
    HDWR_TURRET_LO_OBJ);
        lonmi_SetLaserAttenToCurrObjAndLine();

    /* no gap!!! */
        if( rt == TRUE)
          {
          for( j= 0, rt = FALSE;  rt == FALSE && j < HOME_XY_TIMEOUT_CT; j++)
            {
            rt =mot_CurrAxisGetOneChar(  &ch);
            if( rt == FALSE  ||  (int)ch != 'F')
```

```
                rt = FALSE;
              }
             StageInfo.bStage = rt;
     if(j >= HOME_XY_TIMEOUT_CT )
     fprintf( stderr, "zStillNotHome,");
             }
          if( rt == TRUE)
             {
             j = HDWR_TURRET_LO_OBJ -1;
             if( j < 0)
               j = 0;
             iJGain = pCfg->StageSpec.JoyStickGain[j];
             mot_SetStgJoyStickGain( eAxisX, iJGain);
             mot_SetStgJoyStickGain( eAxisY, iJGain);
             iJGain = pCfg->StageSpec.JoyStickZGain[j];
             mot_SetStgJoyStickGain( eAxisZ, iJGain);
             stg_SetZJoystickLimitToCurrParms();          /* cfg/laserParms/obj set already */
             }
          dat_SendInfo_ObjChanged();

          }
       rt = StageInfo.bStage;
       return (rt );
       }
     /*****************************************************
     * Put Stage Offline
     * return TRUE if online
     *        FALSE if offline
     *****************************************************
     */
     extern BOOL stg_PutStageOffline( void)
       {
       BOOL      rt = FALSE;

       if( StageInfo.bStage == TRUE)
         {
         if( StageInfo.StgStatus  == eStgIdle )
```

```
            {
            rt = mot_PutStageOffline();
            if( rt == TRUE)
              StageInfo.bStage  = FALSE;
            }
          }
        rt = StageInfo.bStage;
        return ( rt);
        }

/***************************************************************
 * home y stage and wait till done
 ***************************************************************
 */
extern BOOL stg_HomeXYAndWaitTillDone( void)
   {
   BOOL       rt, bMoving;
   BYTE            ch;
   int        iRetry;

   rt = mot_MoveToHome( eAxisX);
   if( rt == TRUE)
     {
     rt = FALSE;
     for( iRetry = 0;rt == FALSE && iRetry < HOME_XY_TIMEOUT_CT; iRetry ++)
       {
       rt =mot_CurrAxisGetOneChar(  &ch);
       if( rt == FALSE || (int)ch != 'F')
            rt = FALSE;
       }
     if( rt == TRUE )                    /* wait till X done */
       {
       rt = mot_MoveToHome( eAxisY);
       if( rt == TRUE)
         {
         stgWaitTillCurrAxisStop( 1);
         rt = mot_EnableJoystick();
```

```
          }
        }
      }

fprintf( stderr, "HomeCt:%d=%d,", iRetry, rt );

  return rt;
  }
/******************************************************************
 * return the current Z position in um
 * Send out WakeUnit if bAxisCmd== TRUE
 ******************************************************************
 */
extern BOOL stg_ReadStageZUm( float *Um, BOOL bAxisCmd)
  {
  BOOL      rt = FALSE;
  int       Pos;
  if( StageInfo.StgStatus  == eStgIdle )
    {
    if( bAxisCmd == TRUE)
      rt = mot_ReportOnePos( eAxisZ, &Pos);
    else
      rt = mot_CurrAxisReportMotorCt( &Pos);
    if( rt == TRUE)
      {
      *Um = Pos * StageInfo.ZMResol;
      if( *Um < 0.0 || (double)*Um > STG_Z_TRAVEL_LIM)
        rt = FALSE;
      }
    }
  return rt;
  }
/******************************************************************
 * move Z stage to specified Location in um
 * Send out WakeUnit if bAxisCmd== TRUE
 ******************************************************************
 */
```

```
extern BOOL stg_MoveStageZUm( float Um. BOOL bAxisCmd)
  {
  BOOL      rt = FALSE:
  int       Pos;
  if( StageInfo.StgStatus  == eStgIdle )
    {
    Pos = (double)Um / StageInfo.ZMResol;
    if( bAxisCmd == TRUE)
      rt = mot_AbsMove( eAxisZ. Pos);
    else
      rt = mot_CurrAxisAbsMove( Pos);
    }
 return rt;
 }

/*******************************************************
 * read real current stage XY position
 * return TRUE if both XY valid
 *******************************************************
 */
extern BOOL stg_ReadStageXYPosition( STAGE_XY *StageXY, STG_MOVE_MODE
MoveMode, int iObj)
  {
  BOOL      rt = FALSE;
  double    dEnc;
  int       RawEnc;
  STAGE_XY      StageLocoXY;

  if( StageInfo.StgStatus  == eStgIdle )
    {
    rt = mot_ReportOneEnc( eAxisX. &dEnc. &RawEnc);
    if( rt == TRUE)
      {
     StageLocoXY.x =stgEncoderCount2UM( eAxisX. dEnc. iObj);
     rt = mot_ReportOneEnc( eAxisY. &dEnc. &RawEnc);
     if( rt == TRUE)
       {
            StageLocoXY.y =stgEncoderCount2UM( eAxisY, dEnc, iObj);
            stgOrthoCorrAndDskToAppl( StageXY, &StageLocoXY, MoveMode );
            }
          }
        }
    return ( rt);
    }
/*********************************************************
 * read real current stage XY position, and motor ct/encoder ct offset
 * return TRUE if both XY valid
 * offsets work this way:
 * ——0——————————40000
 * ——————————————20030
 *
 * Moffset = -60 motor ct, assume 1 encoder ct = 2 motor cts.
 * to go to target encoder ct = 20000, initial motor ct = 40000
 *     real motor ct to get to target = 40000 - 60
 *
 *********************************************************
 */
static BOOL stg_RdStgXYPosAndMotorEncoderOffset( STAGE_XY *StageXY,
STG_MOVE_MODE MoveMode,
                                    int iObj,
                                    int *XMotorCtOffset, int *YMotorCtOffset)
  {
  BOOL      rt = FALSE;
  double  dEnc;
  int         RawEnc, MotCt, ConvertedMCt;
  STAGE_XY      StageLocoXY;

  if( StageInfo.StgStatus  == eStgIdle )
    {
    rt = mot_ReportOneEnc( eAxisX, &dEnc, &RawEnc);
    if( rt == TRUE)
       rt = mot_CurrAxisReportMotorCt( &MotCt);
    if( rt == TRUE)
       {
            ConvertedMCt = RawEnc * StageInfo.XEResol/ StageInfo.XMResol;
            *XMotorCtOffset = MotCt - ConvertedMCt;
           StageLocoXY.x =stgEncoderCount2UM( eAxisX. dEnc. iObj);
           rt = mot_ReportOneEnc( eAxisY. &dEnc. &RawEnc);
            }
         if( rt == TRUE)
            rt = mot_CurrAxisReportMotorCt( &MotCt);
         if( rt == TRUE)
            {
            ConvertedMCt = RawEnc * StageInfo.YEResol/ StageInfo.YMResol;
            *YMotorCtOffset = MotCt - ConvertedMCt;
           StageLocoXY.y =stgEncoderCount2UM( eAxisY. dEnc. iObj);
           stgOrthoCorrAndDskToAppl( StageXY. &StageLocoXY. MoveMode );
            }
         }
      return ( rt);
      }

    /****************************************************
     * Move stage with endpoint correction. or stopped by other cmd
     ****************************************************
     */
    static BOOL stg_MoveStageTillDone( STAGE_XY *StageXY. STG_MOVE_MODE MoveMode.
    int iObj,
                                       int XMOffset. int YMOffset)
      {
      int      x, y;
      BOOL   rt= FALSE;
      int  PIDStat. Enc;
      STAGE_STATUS *pStgInfo = &StageInfo;

    fprintf( stderr. "MoveTillDone=");

      if( StageInfo.bStage  ==  TRUE)
          {
          stgOrthoCorrAndDskToStg( &(StageInfo.TargetXY). StageXY. MoveMode );
         x = stgUM2MotorCount( eAxisX. StageInfo.TargetXY.x. iObj);
```

RECTIFIED SHEET (RULE 91)

```
        y = stgUM2MotorCount( eAxisY. StageInfo.TargetXY.y. iObj);

        rt =  mot_AbsXYMove( x + XMOffset. y + YMOffset);
        if( rt == TRUE)
          rt = stgBackgroundProc( pStgInfo. iObj);
        }
fprintf( stderr. "%d.". rt);
  return rt;
  }

/****************************************************
 * background process:
 * do backlash/inaccuracy correction
 *
 * do just one command: wait till xy move done
 * !! assumes axis currently selected is Y
 *
 ****************************************************
 */
static BOOL stgBackgroundProc(STAGE_STATUS *pStgInfo. int iObj)
  {
  char chBuf[30];
  BOOL  rt = FALSE. bMoving;
  int  iRetry;

  for( iRetry = 0;rt == FALSE && iRetry < 4; iRetry ++)
    {
    rt =mot_CurrAxisGetOneChar(  (BYTE *)chBuf);
    if( rt == TRUE)
      if( (int)chBuf[0] != 'F')
        {
        rt = FALSE;
        DebugPrtCh( chBuf[0] );
        }
    }
  if( rt == TRUE && (int)chBuf[0] == 'F')   /* wait till Y done */
     {
        rt = mot_SwitchAxis( eAxisX);
        stgWaitTillCurrAxisStop( 2);

        if( rt == TRUE)
          rt = stgEndPtPosCorrection( eAxisX, pStgInfo->TargetXY.x, iObj,
                            StageInfo.XEResol, StageInfo.XMResol);

        if( rt == TRUE)
          rt = mot_SwitchAxis( eAxisY);

        if( rt == TRUE)
          rt = stgEndPtPosCorrection( eAxisY, pStgInfo->TargetXY.y, iObj,
                            StageInfo.YEResol, StageInfo.YMResol);
        }
    return rt;
    }

/*******************************************************
 * Assume the last move had moved stage to 2 * backlash pass
 * the desirable position.
 * first move back to just one time backlash to fix the backlash
 * problem. Then move an amount base on encoder count to fix
 * the inaccuracy problem.
 *******************************************************
 */

static BOOL stgEndPtPosCorrection(MOT_AXIS Axis, double TargetUM, int iObj,
                            float EResol, float MResol)
 {
 BOOL        rt = FALSE, bDone;
 int      TargetMotCt, RawEnc, iLoopCt;
 double   dEnc, dTargetEnc, dDeltaEnc1;

 dTargetEnc = stgUM2EncoderCount( Axis, TargetUM, iObj);

 bDone = FALSE;
 rt   = TRUE;
```

```
    util_sginap( 4);
    for( iLoopCt = 0; iLoopCt < NUM_CLOSED_LOOPS && bDone == FALSE && rt ==
  TRUE; iLoopCt++)
      {
      rt = mot_CurrAxisReportEnc( &dEnc, &RawEnc);
      if( rt == TRUE)
        {
        dDeltaEnc1 = dTargetEnc - dEnc;
        if( abs( dDeltaEnc1) > MAX_NO_MOVE_ECT)
          {
          TargetMotCt = dDeltaEnc1 * EResol / MResol;
          rt = mot_CurrAxisRelMove( TargetMotCt);
          if( rt == TRUE)
            {
            stgWaitTillCurrAxisStop( 3);
            util_sginap( 4);
            }
          }
        else
          bDone = TRUE;
        }
      }
   return rt;
   }
 /*********************************************************************
  * MoveXYStage
  * send move command to xy-stage
  * but first call MoveZStage to move z-stage away from objective
  *********************************************************************
  */
extern BOOL stg_SafeMoveRawXYStage( STAGE_XY *StageXY, int iObj,
STG_MOVE_F_MODE eFMode)
   {
   BOOL      rt = FALSE;

   rt = stgSafeMoveXYStageRawAndNormal( StageXY, iObj, eFMode, eStgRawPos,
eMoveEnableJoystick);
```

```
  return rt;
  }
/*******************************************************************
 * MoveXYStage
 * send move command to xy-stage
 * but first call MoveZStage to move z-stage away from objective
 *******************************************************************
 */
extern BOOL stg_SafeMoveXYStage( STAGE_XY *StageXY, int iObj, STG_MOVE_F_MODE
eFMode )
  {
  BOOL rt = FALSE;

fprintf( stderr, "StgMove,");
  rt = stgSafeMoveXYStageRawAndNormal( StageXY, iObj, eFMode, eStgNormalPos,
eMoveEnableJoystick);
  return rt;
  }

#define WAVE_LEN_UM          .45
/*******************************************************************
 * Safe move stage, with AutoFine Focus, normal/raw moves
 * if AF is specified: assume stage tilt is in .uvconfig
 *
 *          check depth-of-focus, do AF if XY movement exceed d-of-foc
 *******************************************************************
 */
static BOOL  stgSafeMoveXYStageRawAndNormal( STAGE_XY *StageXY, int iObj,
                         STG_MOVE_F_MODE eFMode, STG_MOVE_MODE MoveMode,
                         STG_MOVE_JOY_STATUS eMoveJoy )
  {
  BOOL           rt;
  STAGE_XY       CurrXY;
  double    deltaX, deltaY;
  int       XMotorCtOffset, YMotorCtOffset;
  float     fNA2, fDepthFoc, fTilt;
  UV_CONFIG      *cfg = cfg_GetLockRdConfig();
```

```
    rt = stg_RdStgXYPosAndMotorEncoderOffset( &CurrXY, eStgRawPos, iObj,
                                      &XMotorCtOffset, &YMotorCtOffset);
    if( rt == TRUE)
      {
      deltaX = CurrXY.x - StageXY->x;
      deltaY = CurrXY.y - StageXY->y;
      if( ( fabs(deltaX) <= MAX_NO_MOVE_UM) && ( fabs(deltaY) <=
  MAX_NO_MOVE_UM) )
         rt = TRUE;
      else
         {
         rt = stg_MoveStageTillDone( StageXY, MoveMode, iObj, XMotorCtOffset,
  YMotorCtOffset);
         if( rt == TRUE)
           {
           switch( eFMode )
             {
             case eStgCoarseAF:
                    lomui_AFAfterXYMove( elomui_CoarseAF);
                    dat_SendInfoToWindows( eFineZPosChanged);
                    break;
             case eStgAutoFocus:
/*
                    fTilt = fTilt * cfg->StageSpec.LevelXOffsetZUm/200000 +
                         (float)deltaY * cfg->StageSpec.LevelYOffsetZUm/200000 ;
*/
                    fTilt = sqrtf( ((double)deltaX * deltaX) + ((double)deltaY * deltaY) );
                    fTilt = (double)fTilt * cfg->StageSpec.LevelXOffsetZUm /200000;
                    fTilt = fabs(fTilt);
                    fNA2 = 1;
                    if( cfg->ObjSpec.NA[iObj] != 0.0)
                      fNA2 = (double)cfg->ObjSpec.NA[iObj] * cfg->ObjSpec.NA[iObj];
                    fDepthFoc = (double)WAVE_LEN_UM / fNA2 /4;
                    if( (double)fTilt > fDepthFoc )
                      {
                      lomui_AFAfterXYMove( elomui_FineAF);
                      dat_SendInfoToWindows( eFineZPosChanged);
```

RECTIFIED SHEET (RULE 91)

```
                    }
                   break;
               }
             if( eMoveJoy != eMoveDisableJoystick)
               mot_EnableXYJoystick();
             }
           }
         }
       return rt;
       }
/**************************************************************
 * Turn on JoyStick
 **************************************************************
 */
extern BOOL stg_EnableJoystick( void)
  {
  return (mot_EnableJoystick()) ;
  }
/*******************************************************
 * stg_relmoveXY() relatively move XY in small distance
 * distance clipped at 100um
 * otherwise no safety check
 * Used for interactive mouse/traceball moves
 *******************************************************
 */
extern BOOL stg_RelMoveXY( int Xum, int Yum, int iObj)
  {
  STAGE_XY      CurrXY;
  BOOL          rt;
  double    ChkLimX, ChkLimY;
  int       XStageDir, YStageDir;

  XStageDir = X_STAGE_DIR;
  YStageDir = Y_STAGE_DIR;

  rt = stg_ReadStageXYPosition( &CurrXY, eStgRawPos, iObj);
  if( rt == TRUE)
```

RECTIFIED SHEET (RULE 91)

```
        {
        ChkLimX = CurrXY.x + Xum;
        ChkLimY = CurrXY.y + Yum;
        if( (ChkLimX * ChkLimX + ChkLimY * ChkLimY) >  STG_SQ_XY_TRAVEL_LIM)
          rt = FALSE;
        else
          {
          if( XStageDir < 0)
            Xum = -Xum;
          if( YStageDir < 0)
            Yum = -Yum;

          rt = mot_RelXYMove( (double)Xum /StageInfo.XMResol,
                              (double)Yum /StageInfo.YMResol );
          if( rt == TRUE)
            {
            stgWaitTillCurrAxisStop( 5);
            rt = mot_SwitchAxis( eAxisX);
            if( rt == TRUE)
              stgWaitTillCurrAxisStop( 6);
            }
          }
        }
      return rt;
      }
/*******************************************************
 * relative move on current axis
 *******************************************************
 */
extern BOOL stg_RelMoveUm(  MOT_AXIS Axis, int iUm, int iObj)
  {
  double      dEnc;
  int         RawEnc, iCurrUm;
  BOOL        rt = FALSE;
  int         XStageDir, YStageDir;

  XStageDir = X_STAGE_DIR;
```

```
    YStageDir = Y_STAGE_DIR;

    rt = mot_ReportOneEnc( Axis, &dEnc, &RawEnc);
    if( rt == TRUE )
      {
      iCurrUm = stgEncoderCount2UM( Axis, dEnc, iObj);
      if( abs( iCurrUm + iUm) > STG_XY_TRAVEL_LIM)
        rt = FALSE;
      else
        {
        if( Axis == eAxisX)
          {
          if( XStageDir < 0)
            iUm = -iUm;
          rt = mot_CurrAxisRelMove( (double)iUm /StageInfo.XMResol);
          }
        else
        if( Axis == eAxisY)
          {
          if( YStageDir < 0)
            iUm = -iUm;
          rt = mot_CurrAxisRelMove( (double)iUm /StageInfo.YMResol);
          }
        else
          rt = FALSE;
        }
      }
    if( rt == TRUE)
      stgWaitTillCurrAxisStop( 7);
    return rt;
    }



/***************************************************
 * set XY axis to default speed
 ***************************************************
 */
extern BOOL stg_SetXYToDefaultSpeeds( void)
  {
  return (mot_SetXYToDefaultSpeeds() );
  }

/************************************************
 * set XY to speed specified
 ************************************************
 */
extern BOOL stg_SetXYSpeedsUm( int iInitSpdUm, int iFinalSpdUm)
  {
  BOOL          rt ;
  int           iInitSpdCt, iFinalSpdCt;

  if( iFinalSpdUm < MIN_XY_SPEED_UM)
    iFinalSpdUm = MIN_XY_SPEED_UM;
  if( iInitSpdUm < 0)
    iInitSpdUm = 0;

  iInitSpdCt  = iInitSpdUm  / StageInfo.XMResol;
  iFinalSpdCt = iFinalSpdUm / StageInfo.XMResol;

  rt = mot_SetInitAndFinalSpeedsCt( eAxisX, iInitSpdCt, iFinalSpdCt);
  if( rt == TRUE)
    {
    iInitSpdCt  = iInitSpdUm  / StageInfo.YMResol;
    iFinalSpdCt = iFinalSpdUm / StageInfo.YMResol;
    rt = mot_SetInitAndFinalSpeedsCt( eAxisY, iInitSpdCt, iFinalSpdCt);
    }
  return rt;
  }

/*******************************************************
 * convert um to MOTOR count
 *******************************************************
 */
```

```
static int stgUM2MotorCount( MOT_AXIS Axis, double um, int iObj)
  {
  int           Counts, iObjOffset;
  double        fOffset, fRes;
  UV_CONFIG      *cfg = cfg_GetLockRdConfig();
  UV_STAGE_SPEC *StageSpec;
  GLOBAL_DATA        *pUV = dat_ObtainGlobDataPtr();
  int           XStageDir, YStageDir, iMapCt;

  XStageDir = X_STAGE_DIR;
  YStageDir = Y_STAGE_DIR;

  StageSpec = &(cfg->StageSpec);
  if( um > STG_XY_TRAVEL_LIM)
    um = STG_XY_TRAVEL_LIM;
  else
    if( um < -STG_XY_TRAVEL_LIM )
      um = -STG_XY_TRAVEL_LIM;

  if( Axis == eAxisY)
      {
      fOffset = YMOT_OFFSET + pUV->UVSysParms.YCenterOffsetUm;
      fRes   = StageInfo.YMResol;
      um = stgIdealUMToMappedUM( Axis, cfg, um);
      if( YStageDir < 0)
        um = -um;
      }
  else
      {
      fOffset = XMOT_OFFSET + pUV->UVSysParms.XCenterOffsetUm;
      fRes   = StageInfo.XMResol;
      um = stgIdealUMToMappedUM( Axis, cfg, um);
      if( XStageDir < 0)
        um = -um;
      }

  Counts = (int) ((um + fOffset)/ fRes);
```

```
   iObjOffset = stgObjMOffsetCt( Axis, StageSpec, iObj);
   Counts += iObjOffset;
   return (Counts);
   }

/***************************************************
 * convert specified( ideal) position to mapped( real stage) position
 ***********************************************************
 */
static double stgIdealUMToMappedUM( MOT_AXIS Axis, UV_CONFIG *pCfg, double
dIdealUM)
   {
   STAGE_MAP_PARMS *pStgMap = &(pCfg->StageMappingParms);
   double     dConverted, dGridLo, dGridHi, dIdealLower;
   int        iCt;

   dConverted = dIdealUM;
   if( Axis == eAxisY)
     {
     if( pStgMap->iYPosCt > 1)
        {
        if( dIdealUM > pStgMap->YGridTable[0].RawPosUm &&
             dIdealUM < pStgMap->YGridTable[pStgMap->iYPosCt -1].RawPosUm )
          {
          iCt = (dIdealUM - pStgMap->YGridTable[0].RawPosUm) /
pStgMap->fYGridSizeUm;
          dIdealLower = pStgMap->YGridTable[ 0 ].RawPosUm + ((double)iCt *
pStgMap->fYGridSizeUm);
          dGridLo    = pStgMap->YGridTable[ iCt    ].RawPosUm;
          dGridHi    = pStgMap->YGridTable[ iCt + 1].RawPosUm;
          dConverted = dGridLo +
                  ( (dIdealUM - dIdealLower) * (dGridHi -dGridLo)/
pStgMap->fYGridSizeUm);
          }
        }
     }
  else
```

```
        {
        if( pStgMap->iXPosCt > 1)
          {
          if( dIdealUM > pStgMap->XGridTable[0].RawPosUm &&
               dIdealUM < pStgMap->XGridTable[pStgMap->iXPosCt -1].RawPosUm )
            {
            iCt = (dIdealUM - pStgMap->XGridTable[0].RawPosUm) /
pStgMap->fXGridSizeUm;
            dIdealLower = pStgMap->XGridTable[ 0 ].RawPosUm + ((double)iCt *
pStgMap->fXGridSizeUm);
            dGridLo   = pStgMap->XGridTable[ iCt].RawPosUm;
            dGridHi   = pStgMap->XGridTable[ iCt + 1].RawPosUm;
            dConverted = dGridLo +
                    ( (dIdealUM - dIdealLower) * (dGridHi -dGridLo)/
pStgMap->fXGridSizeUm);
            }
          }
        }
   return dConverted;
   }

/**********************************************************
 * find the XY offset in motor ct due to Object changes
 **********************************************************
 */
static int stgObjMOffsetCt( MOT_AXIS Axis, UV_STAGE_SPEC *StageSpec, int iObj)
  {
  int  iOffset = 0;
  if( iObj >= 0 && iObj < CONFIG_NUM_OBJ)
    {
    if( Axis == eAxisX)
      iOffset = StageSpec->XOffsetMCt[iObj];
    else
      iOffset = StageSpec->YOffsetMCt[iObj];
    }
  return iOffset;
  }
```

```
/**********************************************************
 * find the XY offset in encoder ct due to Object changes
 **********************************************************
 */
static int stgObjEOffsetCt( MOT_AXIS Axis, UV_STAGE_SPEC *StageSpec, int iObj)
  {
  int  iOffset = 0;
  if( iObj >= 0 && iObj < CONFIG_NUM_OBJ)
    {
    if( Axis == eAxisX)
      iOffset = StageSpec->XOffsetECt[ iObj];
    else
      iOffset = StageSpec->YOffsetECt[ iObj];
    }
  return iOffset;
  }

/**********************************************************
 * convert ENCODER count to um
 **********************************************************
 */
static double stgEncoderCount2UM( MOT_AXIS Axis, double dCounts, int iObj)
  {
  double       UM,  fOffset, fRes;
  int          Dir = 1, iObjOffset;
  UV_CONFIG      *cfg = cfg_GetLockRdConfig();
  UV_STAGE_SPEC *StageSpec;
  GLOBAL_DATA          *pUV = dat_ObtainGlobDataPtr();

  StageSpec = &(cfg->StageSpec);
  iObjOffset = stgObjEOffsetCt( Axis, StageSpec, iObj);
  dCounts -= iObjOffset;
  if( Axis == eAxisY)
     {
     fOffset = YMOT_OFFSET + pUV->UVSysParms.YCenterOffsetUm;
     fRes   = StageInfo.YEResol;
     Dir    = Y_STAGE_DIR;
```

```
        }
    else
        {
        fOffset = XMOT_OFFSET + pUV->UVSysParms.XCenterOffsetUm;
        fRes    = StageInfo.XEResol;
        Dir     = X_STAGE_DIR;
        }
    UM = (dCounts * fRes) - fOffset;
    if( Dir < 0)
      UM = -UM;
    return UM;
    }
/**********************************************************
 * convert um to Encoder count
 **********************************************************
 */
static int stgUM2EncoderCount( MOT_AXIS Axis, double um, int iObj)
    {
    int         Counts, iObjOffset;
    double      fOffset, fRes;
    UV_CONFIG *cfg = cfg_GetLockRdConfig();
    UV_STAGE_SPEC *StageSpec;
    GLOBAL_DATA        *pUV = dat_ObtainGlobDataPtr();
    int         XStageDir, YStageDir;

    XStageDir = X_STAGE_DIR;
    YStageDir = Y_STAGE_DIR;

    StageSpec = &(cfg->StageSpec);
    if( um > STG_XY_TRAVEL_LIM)
      um = STG_XY_TRAVEL_LIM;
    else
      if( um < -STG_XY_TRAVEL_LIM )
        um = -STG_XY_TRAVEL_LIM;

    if( Axis == eAxisY)
        {
```

RECTIFIED SHEET (RULE 91)

```
        fOffset = YMOT_OFFSET + pUV->UVSysParms.YCenterOffsetUm;
        fRes    = StageInfo.YEResol;
        if( YStageDir < 0)
          um = -um;
        }
    else
        {
        fOffset = XMOT_OFFSET + pUV->UVSysParms.XCenterOffsetUm;
        fRes    = StageInfo.XEResol;
        if( XStageDir < 0)
          um = -um;
        }

    Counts = (int) ((um + fOffset)/ fRes);
    iObjOffset = stgObjEOffsetCt( Axis, StageSpec, iObj);
    Counts += iObjOffset;
    return (Counts);
    }


  /*********************************************************************
   * move XY relative ,
   * to compensate for objective changes
   *********************************************************************
   */
  extern BOOL stg_MoveXYToCompensateObjChanges( int iOldObj, int iNewObj)
    {
    int  iOldCt, iNewCt, iXTargetCt, iYTargetCt;
    UV_CONFIG *cfg = cfg_GetLockRdConfig();
    BOOL      rt = FALSE;

    iXTargetCt = 0;
    iYTargetCt = 0;
    if( iOldObj >=0 && iNewObj >=0 && iOldObj < CONFIG_NUM_OBJ && iNewObj <
  CONFIG_NUM_OBJ)
      {
      iOldCt = cfg->StageSpec.XOffsetMCt[ iOldObj];
```

```
        iNewCt = cfg->StageSpec.XOffsetMCt[ iNewObj];
        iXTargetCt = iNewCt - iOldCt;
        iOldCt = cfg->StageSpec.YOffsetMCt[ iOldObj];
        iNewCt = cfg->StageSpec.YOffsetMCt[ iNewObj];
        iYTargetCt = iNewCt - iOldCt;
        rt = mot_RelXYMove( iXTargetCt, iYTargetCt);
        }
    return rt;
    }


  /*******************************************************
   * move stage within the robot's range
   * z should be home as part of the safe procedure of known good Z pos
   *
   * but some control takes extra long( after any actual motion) to home
   * it is takenout for now,
   * but then the motor slip, dangerous, put back in.
   *******************************************************
   */
  extern BOOL stg_MoveCloseForRobot( STG_MOVE_JOY_STATUS eMoveJoy)
    {
    STAGE_XY        stagexy_data;
    BOOL            rt = FALSE;
    int         iRetry;
    BYTE        ch;
    int         XStageDir, YStageDir;
    GLOBAL_DATA         *pUV = dat_ObtainGlobDataPtr();

    XStageDir = X_STAGE_DIR;
    YStageDir = Y_STAGE_DIR;

    /* home z*/
    if( XStageDir >= 0)
      stagexy_data.x =      ROBOT_LOAD_XPOS - pUV->UVSysParms.XCenterOffsetUm;
    else
      stagexy_data.x =      ROBOT_LOAD_XPOS + pUV->UVSysParms.XCenterOffsetUm;
```

```
        if( YStageDir >= 0)
          stagexy_data.y =        ROBOT_LOAD_YPOS - pUV->UVSysParms.YCenterOffsetUm;
        else
          stagexy_data.y =        ROBOT_LOAD_YPOS + pUV->UVSysParms.YCenterOffsetUm;

        rt = stg_HomeZAndWaitTillDone();
        if( rt == TRUE)
          {
          mot_EnableCurrentAxisJoystick();
          rt = stgSafeMoveXYStageRawAndNormal( &stagexy_data, OBJ_MOVE_CLOSE_ROBOT,
                                      eStgNoAF, eStgRawPos, eMoveJoy);


          }
        return rt;
    }
    /******************************************************
     * move z to home and wait till done, or timeout
     ******************************************************
     */
    extern BOOL stg_HomeZAndWaitTillDone( void)
      {
      BOOL      rt = FALSE;
      int       iRetry;
      BYTE            ch;

      if( mot_MoveToHome( eAxisZ) == TRUE)
        {
        for( iRetry = 0; rt == FALSE && iRetry < HOME_XY_TIMEOUT_CT; iRetry ++)
          {
          rt =mot_CurrAxisGetOneChar(  &ch);
          if( rt == FALSE || (int)ch != 'F')
               rt = FALSE;
          }
        }
    if( iRetry > 1)
      fprintf( stderr, "homeZ.NumRetry =%d,", iRetry);
```

```
   return rt;
   }

/*****************************************************
 * move stage within the robot's range
 *****************************************************
 */
extern BOOL stg_IsStageCloseForRobot( void)
{
   int  rt = FALSE;
   return( TRUE);
}
/*****************************************************
 * relative move Z in um
 * only move within limit
 *****************************************************
 */
extern BOOL stg_SafeRelMoveStageZUm( int iUm )
   {
   UV_CONFIG      *pCfg  = cfg_GetLockRdConfig();
   OPTICS_PARMS *pOptic = da_GlobalLonPointer();
   BOOL           rt = FALSE;
   float          fZPos, fZLimUm, fTarget;
   int            iObj;

   lon_APIQuery( eLonAPITurretPosCmd,   &iObj,          0);
   iObj –;
   if( iObj < 0)
     iObj = 0;
   if( iObj >= CONFIG_NUM_OBJ)
     iObj = CONFIG_NUM_OBJ -1;

   fZLimUm    = pCfg->StageSpec.ZLimitUm[iObj] - pOptic->SampleThick;
   if( stg_ReadStageZUm( &fZPos, TRUE) == TRUE)
     {
     fTarget = fZPos + (double)iUm;
     if( fTarget > 0.0 &&  (double)fTarget < fZLimUm)
```

```
            {
            rt = stg_MoveStageZUm( fTarget, TRUE);
            mot_EnableCurrentAxisJoystick();
            }
        }
    return rt;
    }
  /*****************************************************
   * set z joystick travel limit
   * based on current obj, sample thickness, cfg limits
   *****************************************************
   */
  extern BOOL stg_SetZJoystickLimitToCurrParms( void )
    {
    UV_CONFIG      *pCfg = cfg_GetLockRdConfig();
    OPTICS_PARMS *pOptic = da_GlobalLonPointer();
    int           iObj;
    float         fZLimUm;
    BOOL              rt;

    lon_APIQuery( eLonAPITurretPosCmd,   &iObj,          0);
    iObj –;
    fZLimUm   = pCfg->StageSpec.ZLimitUm[iObj] - pOptic->SampleThick;
    rt = mot_SetStgPositiveLim( eAxisZ, (int)( (double)fZLimUm /
  pCfg->StageSpec.ZMStep2um)) ;
    return rt;
    }




  static BOOL DebugPrtCh( char ch)
    {
    int iCh;
    iCh = ch;
    if( iCh > ' ' && iCh < 0x7f)
      fprintf( stderr, "%c", ch);
    else
      fprintf( stderr, "<%x>", iCh);
    return TRUE;
    }


/******************************************
 * check move
 * code is used only to identify caller, each call
 * should have an unique number
 ******************************************
 */
extern BOOL stgWaitTillCurrAxisStop( int Code)
  {
  int iRetry;
  BOOL      bMoving;

  for( bMoving = TRUE, iRetry = 0; iRetry < 200 && bMoving == TRUE; iRetry++)
    {
    bMoving =  mot_IsMotorMoving();
    if( bMoving == TRUE)
       util_sginap( RD_DELAY_TICK );
fprintf( stderr, "ChkMove%d=%d,", Code, bMoving );
    }
  return TRUE;
  }
/***********************************************
 * correct for stage orthogonality
 * from encoder to application
 ***********************************************
 */
static BOOL stgOrthoCorrAndDskToAppl( STAGE_XY *DestXY, STAGE_XY *SrcXY,
STG_MOVE_MODE MoveMode)
  {
  STAGE_XY LocXY;
  UV_CONFIG     *pCfg     = cfg_GetLockRdConfig();
  LocXY.y = SrcXY->y;
  LocXY.x = SrcXY->x - (SrcXY->y *  pCfg->StageSpec.stgOrthoTangent);

  if( MoveMode == eStgNormalPos)
    dsk_InvDeskew( DestXY, &LocXY );
  else
    memcpy(DestXY, &LocXY, sizeof( STAGE_XY) );
  return TRUE;
  }

/**************************************************
 * correct for stage orthogonality
 * and deskew from appl to motor
 **************************************************
 */
static BOOL stgOrthoCorrAndDskToStg( STAGE_XY *DestXY, STAGE_XY *SrcXY,
STG_MOVE_MODE MoveMode )
  {
  STAGE_XY      LocXY;
  UV_CONFIG     *pCfg    = cfg_GetLockRdConfig();


  LocXY.y = SrcXY->y;
  LocXY.x = SrcXY->x + (SrcXY->y *  pCfg->StageSpec.stgOrthoTangent);

  if( MoveMode == eStgNormalPos)
    dsk_Deskew( &LocXY, DestXY);
  else
    memcpy( DestXY, &LocXY, sizeof( STAGE_XY) );
  return TRUE;
  }
□
```

```
/****************************************************
 * (c) Copyright 1992,1993, Ultrapointe Corp.
 * All rights reserved
 *
 * motor.c
 *
 * UltraView1000 software
 * this module handles the low level motor interface:
 *
 ****************************************************
 */
#ifdef Irix5plus
#define NeedFunctionPrototypes 1
#else
#define FUNCPROTO
#endif

#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include "uv_util.h"
#include "uvconfig.h"
#include "motor.h"
#include "rs232.h"
#include "lon_msg.h"
#include "lon_img.h"
#include "options.h"

#define TIMEOUT_LINE   5      /*timeout: 5second for line */
#define TIMEOUT_MOV_DONE 4
#define TIMEOUT_CHAR   2

#define MAX_MOTOR_SPEED  150000
#define DEFAULT_ACC     240000      /* 7um/sec/sec */
#define START_MTR_SPEED   2500      /* um/sec  */

#define MAX_ZMOTOR_SPEED  3000
```

```
#define START_ZMTR_SPEED   300          /* this is important for AutoFocus */
#define DEFAULT_ZACC   20000      /* um/sec/sec */

#define XY_POSITIVE_LIM_UM   230000
#define Z_POSITIVE_LIM_UM       8000

#define CH_BUFFER_SIZE 80
static HANDLE               hdStageX = -1;
static HANDLE               hdStageY = -1;
static HANDLE               hdStageZ = -1;
static RS232_PORT_SELECT SGIPortX;          /* GMI_RS2_0, 1, SGI */
static RS232_PORT_SELECT SGIPortY;
static RS232_PORT_SELECT SGIPortZ;

static MOT_AXIS     CurrAxis = eAxisNone;
static RS232_PORT_SELECT CurrSGIPort;
static HANDLE               CurrhdStage;


static BOOL         bXOnline = FALSE;
static BOOL         bYOnline = FALSE;
static BOOL         bZOnline = FALSE;

#define CMD_ABS_MOVE                "MA%d\r"
#define CMD_REL_MOVE                "MR%d\r"
#define CMD_MOVE_HOME               "MH\r"
#define CMD_REP_POS            "RP\r\n"
#define CMD_REP_ENC            "RE\r\n"
#define CMD_SET_SPEED               "VF%d\r"
#define CMD_SET_INIT_SPEED  "VI%d\r"
#define CMD_SET_ACC            "AC%d\r"
#define CMD_SET_POSITION       "SP%d\r"
#define CMD_SET_ENCODER             "SE%d\r"
#define CMD_WAKE_X             "WU1\r"
#define CMD_WAKE_Y             "WU2\r"              /* some delay */
#define CMD_WAKE_Z             "WU3\r"
#define CMD_WAKE_NOBODY             "WU9\r"
```

```
#define CMD_SHORT_CMD            "MED\r"
#define CMD_STOP            "ST\r"
#define CMD_LO_IDLE            "ICD\r"
#define CMD_ENABLE_JOY            "AE\r "
#define CMD_DISABLE_JOY            "\33\r"
#define CMD_JOYSTICK_GAIN  "JG%d\r"
#define CMD_JOY_DEADBAND  "JD15\r"
#define CMD_POSITIVE_LIM   "YP%d\r"
#define CMD_NEGATIVE_LIM   "YN%d\r"

#define CMD_MOV_DONE_ACK "MFE\r"
#define CMD_MOV_DONE_NAK        "MFD\r"
#define CMD_REPORT_STATUS "RS\r"
#define CMD_NOTHING            "\r"  /* delay, space filler */
#define EOL                  0xd

static BOOL motProcessReportMessage( int *Pos);
static BOOL motOnlineOneAxisInit( HANDLE hdStage, RS232_PORT_SELECT SGIPort,
                        int InitSpd, int FinalSpd, int Acc, int PosLimUm, float MStep2um);
static BOOL motInitAccSpeed( int InitSpd, int FinalSpd, int Acc, float MStep2um);
static BOOL motWakeupAxisAndCmd(MOT_AXIS Axis, char *pCmd);
static BOOL motWakeupAxisFast(MOT_AXIS Axis);
static BOOL motCurrAxisSendStringAndGetLine( char *Cmd, char *chBuff, int iSize);
static BOOL motNoCheckWakeUnit( HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*WakeCmd);
static void DebugDisplayOneChar( BYTE  chBuf);
static BOOL motSendAndWaitStar( HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*pCmd);
static BOOL motCurrAxisSendCmdToReadInt( char *pQuery, int *iValue);
static double  motMapEncCt( int iNumPos, double dGridSizeEnc,
                              STAGE_MAP_GRID_PAIR *GridTable, int iRawCt);
static BOOL motCurrAxisRdMappedEncCt( double *dEncCt, int *iRawCt);



/******* these function is for NEAT compatibility **************/
static BOOL command_noecho(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*cmdstr);
static BOOL command_noresp(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*cmdstr);
static RS2_CMD_STATUS   send_echostr(HANDLE hdStage, RS232_PORT_SELECT SGIPort,
char *str);
static BOOL wake_unit_funct_fast(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*WakeCmd);

/**************************************************************/


/*****************************************************
 * initialization once per program
 *****************************************************
 */
extern BOOL mot_Initialize( int      hUldd)               /* program start */
  {
  hdStageZ = hUldd;
  hdStageX = hUldd;
  hdStageY = hUldd;
  return TRUE;
  }
extern BOOL mot_ClosePort( void)                  /* program end */
  {
  mot_DisableJoystick();
  rs2_ClosePort(hdStageZ, SGIPortZ );
  if( hdStageZ != hdStageX)
    {
    rs2_ClosePort(hdStageX, SGIPortX );
    rs2_ClosePort(hdStageY, SGIPortY );
    }
  hdStageX = -1;
  hdStageY = -1;
  hdStageZ = -1;
  return TRUE;
  }
/**************************************************
```

```
 * user turn on stage
 **************************************************
 */
extern BOOL mot_PutStageOnline( void)
  {
  UV_CONFIG      *pCfg = cfg_GetLockRdConfig();
  UV_STAGE_SPEC *StageSpec;
  BOOL             rt = FALSE;
  char        chBuf[CH_BUFFER_SIZE + 10];
  volatile int j;

  if( opt_HasOptionHardware() != TRUE)
    return FALSE;

  StageSpec = &(pCfg->StageSpec);

  if( StageSpec->PortNum >= 0 && StageSpec->PortNum < 6 )
    {
    switch( StageSpec->PortNum)
      {
      case 0:
      case 1: SGIPortX = eSGI_RS2_Port;
              SGIPortY = eSGI_RS2_Port;
              SGIPortZ = eSGI_RS2_Port;
              break;

      case 2: SGIPortX = eGMI_RS2_0;
              SGIPortY = eGMI_RS2_0;
              SGIPortZ = eGMI_RS2_0;
              break;

      case 4:
      case 5:
              SGIPortX = eSGI_RS2_Port;
              SGIPortY = eSGI_RS2_Port;
              SGIPortZ = eGMI_RS2_1;
              break;
```

```
            case 3:
            default:SGIPortX = eGMI_RS2_1;
                  SGIPortY = eGMI_RS2_1;
                  SGIPortZ = eGMI_RS2_1;
                  break;
            }
         if( StageSpec->PortNum == 4 || StageSpec->PortNum == 5)
            {
            hdStageZ  = rs2_initialize( hdStageZ, 3, BAUDRATE96);
            if( StageSpec->PortNum == 4)
              {
              hdStageX  = rs2_initialize( hdStageZ, 0, BAUDRATE96);
              hdStageY  = rs2_initialize( hdStageZ, 1, BAUDRATE96);
              }
            else
              {
              hdStageX  = rs2_initialize( hdStageZ, 10, BAUDRATE96);
              hdStageY  = rs2_initialize( hdStageZ, 11, BAUDRATE96);
              }
            }
         else
            {
            hdStageZ  = rs2_initialize( hdStageZ, StageSpec->PortNum, BAUDRATE96);
            hdStageX  = hdStageZ;
            hdStageY  = hdStageZ;
            }
    fprintf( stderr, "initialized axisxyz=%x,%x,%x,", hdStageX, hdStageY, hdStageZ );

    /********
     * all these extra wakeups seem neccessary to ensure
     * reliable init, especially at controller powe-up
     ********
     */
         bXOnline = bYOnline = bZOnline = FALSE;
         if( StageSpec->bZOnline == TRUE)
            mot_SendWakeUnitString( eAxisZ);        /* z axis is very difficult to talk to */
```

```
          if( StageSpec->bXOnline == TRUE)
            {
            rt = motNoCheckWakeUnit( hdStageX, SGIPortX, CMD_WAKE_X);
            if( rt == TRUE)
              rt = motOnlineOneAxisInit( hdStageX, SGIPortX,
                                      StageSpec->iXYInitSpdUm, StageSpec->iXYFinalSpdUm,
                                      StageSpec->iXYAccUm,
                                      XY_POSITIVE_LIM_UM, StageSpec->XMStep2um);
            bXOnline = rt;
            }
          if( StageSpec->bYOnline == TRUE && rt == TRUE)
            {
            rt = motNoCheckWakeUnit( hdStageY, SGIPortY, CMD_WAKE_Y);
            if( rt == TRUE)
              rt = motOnlineOneAxisInit( hdStageY, SGIPortY,
                                      StageSpec->iXYInitSpdUm, StageSpec->iXYFinalSpdUm,
                                      StageSpec->iXYAccUm,
                                      XY_POSITIVE_LIM_UM, StageSpec->YMStep2um);
            bYOnline = rt;
            }
          if( StageSpec->bZOnline == TRUE && rt == TRUE )
            {
            rt = motNoCheckWakeUnit( hdStageZ, SGIPortZ, CMD_WAKE_Z);
            if( rt == TRUE)
              rt = motOnlineOneAxisInit( hdStageZ, SGIPortZ,
                                      START_ZMTR_SPEED ,MAX_ZMOTOR_SPEED,
                                      DEFAULT_ZACC,
                                      StageSpec->ZLimitUm[0], StageSpec->ZMStep2um);
            bZOnline = rt;
            }
      fprintf( stderr, "XYZonline=%d,%d,%d,", bXOnline, bYOnline, bZOnline);
        if( rt == TRUE)
          rt = (bXOnline || bYOnline || bZOnline);
        }
      return rt;
      }
```

```
/*********************************************
 * initialize one axis
 *********************************************
 */
static BOOL motOnlineOneAxisInit( HANDLE hdStage, RS232_PORT_SELECT SGIPort,
                         int InitSpd, int FinalSpd, int Acc, int PosLimUm, float MStep2um)
{
BOOL     rt;
char chBuf[100];

rt = motSendAndWaitStar( hdStage, SGIPort, CMD_SHORT_CMD);
if( rt == TRUE)
  rt = motSendAndWaitStar( hdStage, SGIPort, CMD_STOP);
if( rt == TRUE)
  {
  sprintf( chBuf, CMD_SET_INIT_SPEED, (int)( (double)InitSpd / MStep2um));
  rt = motSendAndWaitStar(hdStage, SGIPort, chBuf );
  }
if( rt == TRUE)
  {
  sprintf( chBuf, CMD_SET_SPEED,  (int)( (double)FinalSpd / MStep2um));
  rt = motSendAndWaitStar(hdStage, SGIPort, chBuf );
  }
if( rt == TRUE)
  {
  sprintf( chBuf, CMD_SET_ACC,  (int)( (double)Acc / MStep2um));
  rt = motSendAndWaitStar( hdStage, SGIPort, chBuf);
  }
if( rt == TRUE)
  rt = motSendAndWaitStar( hdStage, SGIPort, CMD_LO_IDLE); /* idle curr diabled */
if( rt == TRUE)
  rt = motSendAndWaitStar( hdStage, SGIPort, CMD_JOY_DEADBAND);
if( rt == TRUE)
  rt = motSendAndWaitStar( hdStage, SGIPort, CMD_MOV_DONE_ACK);
if( rt == TRUE)
  rt = mot_SetCurrentStgNegativeLim( 0);
if( rt == TRUE)
```

```
    rt = mot_SetCurrentStgPositiveLim(  (double)PosLimUm / MStep2um );
  if( rt == TRUE)
    rt = mot_SetCurrentStgJoyStickGain( 1);
  return rt;
  }
/***************************************************
 * set XY axis to default speed
 ***************************************************
 */
extern BOOL mot_SetXYToDefaultSpeeds( void )
  {
  char              chBuf[100];
  UV_CONFIG         *cfg = cfg_GetLockRdConfig();
  BOOL              rt = FALSE;
  float        XMStep2Um, YMStep2Um;

  XMStep2Um =  cfg->StageSpec.XMStep2um;
  YMStep2Um =  cfg->StageSpec.YMStep2um;

  rt = motWakeupAxisAndCmd( eAxisX, NULL);
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_SET_INIT_SPEED,  (int)( (double)cfg->StageSpec.iXYInitSpdUm/
XMStep2Um) );
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf);
    }
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_SET_SPEED,  (int)( (double)cfg->StageSpec.iXYFinalSpdUm/
XMStep2Um) );
    rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf);
    }
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisY, NULL);
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_SET_INIT_SPEED,  (int)( (double)cfg->StageSpec.iXYInitSpdUm/
YMStep2Um) );
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf);
    }
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_SET_SPEED,  (int)( (double)cfg->StageSpec.iXYFinalSpdUm/
YMStep2Um) );
    rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf);
    }
  return rt;
  }


extern BOOL mot_PutStageOffline( void)
  {
  mot_DisableJoystick();
  if( bZOnline)
    rs2_ClosePort(hdStageZ, SGIPortZ );
  if( hdStageZ != hdStageX && bXOnline && bYOnline)
    {
    rs2_ClosePort(hdStageX, SGIPortX );
    rs2_ClosePort(hdStageY, SGIPortY );
    }
  CurrhdStage = -1;
  bXOnline = FALSE;
  bYOnline = FALSE;
  bZOnline = FALSE;
  return TRUE;
  }

/********************************************************
 * high level functions
 ********************************************************
 */
extern BOOL mot_RelMove(MOT_AXIS Axis, int Pos)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_REL_MOVE, Pos);
  rt = motWakeupAxisAndCmd( Axis, chBuf);
  return rt;
  }
extern BOOL mot_CurrAxisRelMove( int Pos)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_REL_MOVE, Pos);
  rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf);
  return rt;
  }


extern BOOL mot_AbsMove(MOT_AXIS Axis, int Pos)
  {
  char chBuf[80];
  BOOL      rt = FALSE ;

  sprintf( chBuf, CMD_ABS_MOVE, Pos);
  rt = motWakeupAxisAndCmd( Axis, chBuf);
  return rt;
  }
/*******************************************************
 * move both XY
 *******************************************************
 */
extern BOOL mot_AbsXYMove( int XPos, int YPos)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_ABS_MOVE, XPos);
  rt = motWakeupAxisAndCmd( eAxisX, chBuf);
```

```
    if( rt == TRUE)
      {
      sprintf( chBuf, CMD_ABS_MOVE, YPos);
      rt = motWakeupAxisAndCmd( eAxisY, chBuf);
      }
    return rt;
    }

/****************************************************
 * this function do relative move,
 * will not wait for stop
 ****************************************************
 */
extern BOOL mot_RelXYMove( int XPos, int YPos)
  {
  char chBuf[80];
  BOOL      rt = FALSE ;

  sprintf( chBuf, CMD_REL_MOVE, XPos);
  rt = motWakeupAxisAndCmd( eAxisX, chBuf);
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_REL_MOVE, YPos);
    rt = motWakeupAxisAndCmd( eAxisY, chBuf);
    }
  return rt;
  }
/****************************************************
 * enable XYZ joystick
 ****************************************************
 */
extern BOOL mot_EnableJoystick( void)
  {
  volatile int j;
  BOOL      rt= FALSE;

  rt = motWakeupAxisAndCmd( eAxisX, CMD_ENABLE_JOY);
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisY, CMD_ENABLE_JOY);
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisZ, CMD_ENABLE_JOY);
  return rt;
  }

/*****************************************************
 * enable XYZ joystick and set xy joystick gain
 *****************************************************
 */
extern BOOL mot_EnableJoystickAndSetXYGain( int iJGain)
  {
  char        chBuf[80];
  BOOL             rt= FALSE;

  if( iJGain < 1)  iJGain = 1;
  if( iJGain > 60) iJGain = 60;

  sprintf( chBuf, CMD_JOYSTICK_GAIN,  iJGain);
  rt = motWakeupAxisAndCmd( eAxisX, chBuf);
  if( rt == TRUE)
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, CMD_ENABLE_JOY);
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisY, chBuf);
  if( rt == TRUE)
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, CMD_ENABLE_JOY);
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisZ, CMD_ENABLE_JOY);
  return rt;
  }

/*****************************************************
 * enable XY joystick
 *****************************************************
 */
extern BOOL mot_EnableXYJoystick( void)
```

```
  {
  volatile int j;
  BOOL     rt= FALSE;

  rt = motWakeupAxisAndCmd( eAxisX, CMD_ENABLE_JOY);
  if( rt == TRUE)
    rt = motWakeupAxisAndCmd( eAxisY, CMD_ENABLE_JOY);
  return rt;
  }
/*****************************************************
* disable joystick
*****************************************************
*/
extern BOOL mot_DisableJoystick( void)
  {
  BOOL     rt = FALSE;
  if(  bXOnline == TRUE)
      {
      if( hdStageZ == hdStageX)
        rt = motSendAndWaitStar( hdStageX, SGIPortX, CMD_WAKE_X);
      else
        rt = TRUE;
      if( rt == TRUE)
        rt = motSendAndWaitStar( hdStageX, SGIPortX, CMD_DISABLE_JOY);
      if( rt == TRUE)
        rt = rs2_WaitForResp( hdStageX, SGIPortX, '*', TIMEOUT_CHAR);
      }
  if(  rt == TRUE && bYOnline == TRUE)
      {
      if( hdStageZ == hdStageX)
        rt = motSendAndWaitStar( hdStageY, SGIPortY, CMD_WAKE_Y);
      if( rt == TRUE)
        {
        rt = motSendAndWaitStar( hdStageY, SGIPortY, CMD_DISABLE_JOY);
        if( rt == TRUE)
          rt = rs2_WaitForResp( hdStageY, SGIPortY, '*', TIMEOUT_CHAR);
        }
```

```
            }
        if(  rt == TRUE && bZOnline == TRUE)
            {
            if( hdStageZ == hdStageX)
              rt = motSendAndWaitStar( hdStageZ, SGIPortZ, CMD_WAKE_Z);
            if( rt == TRUE)
              {
              rt = motSendAndWaitStar( hdStageZ, SGIPortZ, CMD_DISABLE_JOY);
              if( rt == TRUE)
                rt = rs2_WaitForResp( hdStageZ, SGIPortZ, '*', TIMEOUT_CHAR);
              }
            }
       return rt;
       }
/*************************************************
 * enable joystick, no checking
 *************************************************
 */
extern BOOL mot_EnableCurrentAxisJoystick( void)
  {
  return( motSendAndWaitStar( CurrhdStage, CurrSGIPort, CMD_ENABLE_JOY) );
  }
/*************************************************
 * stop current axis
 * need to send without normal char-by-char echo check
 * because we may get the 'F'  status in the middle of sending
 * the stop cmd
 *************************************************
 */
extern BOOL mot_StopCurrentAxis( void)
  {
  BOOL rt;

  rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, CMD_STOP);
  return rt;
  }
```

```
/****************************************************************
 * send command to home stage
 ****************************************************************
 */
extern BOOL mot_MoveToHome(MOT_AXIS Axis)
  {
  BOOL      rt = FALSE;

  rt = motWakeupAxisAndCmd( Axis, CMD_STOP);
  if( rt == TRUE)
    rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, CMD_MOVE_HOME);
  return rt;
  }
/****************************************************************
 * send command to home stage
 ****************************************************************
 */
extern BOOL mot_CurrAxisMoveHome( void )
  {
  BOOL      rt = FALSE;

  rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, CMD_MOVE_HOME);
  return rt;
  }

/*****************************************************
 * Send Stop Command to the specified axis
 *****************************************************
 */
extern BOOL mot_StopOneAxis(MOT_AXIS Axis)
  {
  BOOL      rt = FALSE ;

  rt = motWakeupAxisAndCmd( Axis, CMD_STOP);
  return rt;
  }
extern BOOL mot_SetPos(MOT_AXIS Axis, int Pos)
```

```
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_SET_POSITION, Pos);
  rt = motWakeupAxisAndCmd( Axis, chBuf);

  return rt;
  }
extern BOOL mot_SetAcc(MOT_AXIS Axis, int Acc)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_SET_ACC, Acc);
  rt = motWakeupAxisAndCmd( Axis, chBuf);
  return rt;
  }
extern BOOL mot_SetEnc(MOT_AXIS Axis, int Pos)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_SET_ENCODER, Pos);
  rt = motWakeupAxisAndCmd( Axis, chBuf);

  return rt;
  }

/************************************************************
 * query position/encoder
 ************************************************************
 */
extern BOOL mot_ReportOnePos(MOT_AXIS Axis, int *Pos)
  {
  BOOL rt = FALSE;
```

```
    *Pos = 0;
    if( motWakeupAxisAndCmd(Axis, NULL) == TRUE)
      rt = motCurrAxisSendCmdToReadInt( CMD_REP_POS, Pos);
    return rt;
    }
  /*******************************************************
   * query current axis motor counts
   *******************************************************
   */
  extern BOOL mot_CurrAxisReportMotorCt(  int *MotCt)
    {
    BOOL      rt = FALSE;

    rt = motCurrAxisSendCmdToReadInt( CMD_REP_POS, MotCt);
    return (rt);
    }
  /*******************************************
   * report encoder count
   *******************************************
   */
  extern BOOL mot_ReportOneEnc(MOT_AXIS Axis, double *Pos, int *RawPos)
    {
    BOOL rt = FALSE;

    if( motWakeupAxisFast( Axis) == TRUE)
      rt = motCurrAxisRdMappedEncCt( Pos, RawPos);
    return rt;
    }
  /*******************************************************
   * report current axis encode count
   *******************************************************
   */
  extern BOOL mot_CurrAxisReportEnc( double *Pos, int *RawPos)
    {
    BOOL rt = FALSE;

    rt = motCurrAxisRdMappedEncCt( Pos, RawPos);
  return rt;
  }
/*************************************************
 * apply stage mapping to encoder read
 * encoder count read from stage is matched to
 *           RawCt in uvconfig,
 * mapped encode cts are from Computed Counts
 *
 * Computed   ——0 ——1———  ——n-1 ——
 * raw Ct    —— 0———1 ——  ——n-1 ——
 *  case 1     *
 *  case 2            *
 *  case 3                   *
 *
 *
 *  No correction done to case 1,3
 *************************************************
 */
static BOOL motCurrAxisRdMappedEncCt( double *dEncCt, int *iRawCt)
  {
  UV_CONFIG     *cfg = cfg_GetLockRdConfig();
  BOOL          rt = FALSE, bFound;
  int       iNumPos, iPosCt;
  double    dGridSizeEnc;
  STAGE_MAP_GRID_PAIR     *GridTable;

  *dEncCt = 0;
  *iRawCt = 0;
  rt= motCurrAxisSendCmdToReadInt( CMD_REP_ENC, iRawCt);
  if( rt == TRUE && (CurrAxis == eAxisX || CurrAxis == eAxisY ) )
    {
    *dEncCt = *iRawCt;
    if( CurrAxis == eAxisX)
      {
      iNumPos     = cfg->StageMappingParms.iXPosCt;
      dGridSizeEnc     = cfg->StageMappingParms.dXGridSizeEncCt;
      GridTable    = cfg->StageMappingParms.XGridTable;
```

```
            }
          else
            {
            iNumPos      = cfg->StageMappingParms.iYPosCt;
            dGridSizeEnc         = cfg->StageMappingParms.dYGridSizeEncCt;
            GridTable    = cfg->StageMappingParms.YGridTable;
            }
          *dEncCt = motMapEncCt( iNumPos, dGridSizeEnc, GridTable, *iRawCt);
          }
        return rt;
        }

      /*********************************************
       * convert raw encoder ct to Mapped Enc ct
       * depending on stage axis direction
       * the array of Counts can be either increasing/
       *      or decreasing
       *
       *********************************************
       */
      static double  motMapEncCt( int iNumPos, double dGridSizeEnc,
                                  STAGE_MAP_GRID_PAIR *GridTable, int iRawCt)
        {
        double      dMappedEnc, dComputedCtLo;
        BOOL                bFound;
        int         iPosCt, iRawCtLo, iRawCtHi;

        dMappedEnc = iRawCt;
        if( iNumPos > 1)
          {
          if( GridTable[0].RawEncCt  < GridTable[1].RawEncCt)   /* normal.. number increasing */
            {
            if( iRawCt > GridTable[0].RawEncCt && iRawCt < GridTable[iNumPos -1].RawEncCt)
              {
              bFound = FALSE;
              for( iPosCt= 1; iPosCt < (iNumPos -1) && bFound == FALSE; )
                {
```

```
            if( iRawCt < GridTable[ iPosCt].RawEncCt)
               bFound = TRUE;
            else
               iPosCt++;
            }

          dComputedCtLo     = GridTable[0].RawEncCt + ((iPosCt-1) * dGridSizeEnc);
          iRawCtLo          = GridTable[ iPosCt -1].RawEncCt;
          iRawCtHi          = GridTable[ iPosCt   ].RawEncCt;
          dMappedEnc = dComputedCtLo + ((double)( iRawCt - iRawCtLo) * dGridSizeEnc /
                                      (iRawCtHi - iRawCtLo));
          }
        }
      else
        {                                             /* decreasing */
        if( iRawCt < GridTable[0].RawEncCt && iRawCt > GridTable[iNumPos -1].RawEncCt)
          {
          bFound = FALSE;
          for( iPosCt= 1; iPosCt < (iNumPos -1) && bFound == FALSE; )
            {
            if( iRawCt > GridTable[ iPosCt].RawEncCt)
               bFound = TRUE;
            else
               iPosCt++;
            }

          dComputedCtLo     = GridTable[0].RawEncCt - ((iPosCt-1) * dGridSizeEnc);
          iRawCtLo          = GridTable[ iPosCt -1].RawEncCt;
          iRawCtHi          = GridTable[ iPosCt   ].RawEncCt;
          dMappedEnc = dComputedCtLo - ((double)( iRawCt - iRawCtLo ) * dGridSizeEnc /
                                      (iRawCtHi - iRawCtLo));
          }
/*
fprintf( stderr, "Ct=%d, Raw/Computed Ct=%d,%d \n", iPosCt, iRawCt, iMappedEnc);
*/
        }
    }
```

```
   return dMappedEnc;
   }


/*******************************************************
 * send cmd(must be a query cmd), and expect to get an integer
 *******************************************************
 */
static motCurrAxisSendCmdToReadInt( char *pQuery, int *iValue)
  {
  BOOL rt = FALSE;
  if( hdStageZ != hdStageX)
    {
    if ( rs2_FlushQueSendString( CurrhdStage, CurrSGIPort, pQuery) == TRUE)
       rt = motProcessReportMessage( iValue);
    }
  else
    {
    if(  rs2_SendStringWithPerCharHandShake(CurrhdStage, CurrSGIPort, pQuery) ==
eRS2StatusOK)
       rt = motProcessReportMessage( iValue);
    }
  return rt;
  }

/**********************************************
 * the report message has the command mixed in with the result
 * assuming the real reports are all in numbers
 ***********************************************************
 */
static BOOL motProcessReportMessage( int *Pos)
  {
  BOOL rt;
  char chBuf[CH_BUFFER_SIZE + 2];
  int j, iLen, Sign= 1;

  rt = rs2_GetOneLine(CurrhdStage, CurrSGIPort, chBuf, CH_BUFFER_SIZE,
 TIMEOUT_LINE);
   if( rt == TRUE)
     rt = rs2_WaitForResp( CurrhdStage, CurrSGIPort, '*', 2);
   iLen = strlen( chBuf);
   if( rt == TRUE && iLen > 0 )
     {
     for( j=0; j < iLen && rt == TRUE && (int)chBuf[j] != EOL; j++)
        if( !isdigit( chBuf[j] ) && !isalpha( chBuf[j]) && chBuf[j] != (unsigned)'-' )
              rt = FALSE;

     for( j=0; j < iLen && !isdigit( chBuf[j]) && chBuf[j] != (unsigned)'-' ; j++)
        {};
     if( j < iLen && rt == TRUE )
        {
        *Pos = atoi( &chBuf[j]);
        rt = TRUE;
        }
     else
        rt = FALSE;
 if( rt == FALSE)
 fprintf( stderr, "str=%s,\n", chBuf);
     }
   return rt;
   }
/*******************************************************************
 * send a string to neat, and get a response
 *******************************************************************
 */
static BOOL motCurrAxisSendStringAndGetLine( char *Cmd, char *chBuff, int iSize)
  {
  BOOL rt = FALSE;
  if( hdStageZ != hdStageX)
    {
    if( rs2_FlushQueSendString( CurrhdStage, CurrSGIPort, Cmd) == TRUE)
       rt = rs2_GetOneLine( CurrhdStage, CurrSGIPort, chBuff, iSize, TIMEOUT_LINE);
    }
  else
```

```
    {
    if( rs2_SendStringWithPerCharHandShake( CurrhdStage, CurrSGIPort, Cmd) ==
eRS2StatusOK)
      rt = rs2_GetOneLine( CurrhdStage, CurrSGIPort, chBuff, iSize, TIMEOUT_LINE);
    }
  if( rt == TRUE)
    rt = rs2_WaitForResp( CurrhdStage, CurrSGIPort, '*', 2);
  return rt;
  }


extern BOOL mot_SetSpeed( MOT_AXIS Axis, int Speed)
  {
  char chBuf[80];
  BOOL      rt = FALSE;

  sprintf( chBuf, CMD_SET_SPEED, Speed);
  rt = motWakeupAxisAndCmd( Axis, chBuf);
  return rt;
  }

/*******************************************
 * set init and final speed to specified axis
 *******************************************
 */
extern BOOL mot_SetInitAndFinalSpeedsCt( MOT_AXIS Axis, int iInitSpdCt, int iFinalSpdCt)
  {
  char chBuf[80];
  BOOL      rt;

  sprintf( chBuf, CMD_SET_INIT_SPEED, iInitSpdCt);
  rt = motWakeupAxisAndCmd( Axis, chBuf);
  if( rt == TRUE)
    {
    sprintf( chBuf, CMD_SET_SPEED,  iFinalSpdCt );
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf );
    }
    return rt;
    }

 extern BOOL mot_SetCurrAxisSpeed( int Speed)
    {
    char chBuf[80];
    BOOL      rt = FALSE;

    sprintf( chBuf, CMD_SET_SPEED, Speed);
    rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf);
    return rt;
    }
/**********************************************************
 * reset speed to default
 * if( bAxisCmd == TRUE, send WakeUnit command
 **********************************************************
 */
extern BOOL mot_SetDefaultSpeed( MOT_AXIS Axis, BOOL bAxisCmd)
    {
    UV_CONFIG     *cfg    = cfg_GetLockRdConfig();
    char chBuf[80];
    BOOL      rt = FALSE;
    int  Speed;

    switch( Axis)
      {
      case eAxisX: Speed = (double)cfg- > StageSpec.iXYFinalSpdUm/
cfg- > StageSpec.XMStep2um;
                 break;
      case eAxisY: Speed = (double)cfg- > StageSpec.iXYFinalSpdUm/
cfg- > StageSpec.YMStep2um;
                 break;

      case eAxisZ:
      default:   Speed = (double)MAX_ZMOTOR_SPEED / cfg- > StageSpec.ZMStep2um;
                 break;
      }
    sprintf( chBuf, CMD_SET_SPEED, Speed);
    if( bAxisCmd == TRUE)
      rt = motWakeupAxisAndCmd( Axis, chBuf);
    else
      rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf);

    return rt;
    }
  /*****************************************************
   * che f the specified axis is moving
   * return TRUE if yes
   *****************************************************
   */
  extern BOOL mot_IsAxisMoving(MOT_AXIS Axis)
    {
    BOOL      rt = FALSE;

    if( motWakeupAxisAndCmd( Axis, NULL) == TRUE)
      rt = mot_IsMotorMoving();
    return (rt);
    }

  /*****************************************************
   * check if the currently selected motor is moving
   * return FALSE if no motor connected
   * NOTE:
   * !!!!!
   * this is control specific. Check here if need to change
   *  controller
   * !!!!!
   *   while Report Status does not return 'F', wait and retry
   *  after max retry, still invalid response, then assume stopped
   ******************************************************************
   */
  extern BOOL mot_IsMotorMoving( void)
    {
    char        chBuf[CH_BUFFER_SIZE + 2];
```

RECTIFIED SHEET (RULE 91)

```
    int          count, j;
    BOOL              rt = FALSE, bSure = FALSE;

    if( bXOnline == TRUE ||  bYOnline == TRUE || bZOnline == TRUE )
      {
      chBuf[0] = 0;
      chBuf[1] = 0;
      chBuf[2] = 0;
      rt = motCurrAxisSendStringAndGetLine( CMD_REPORT_STATUS, chBuf,
  CH_BUFFER_SIZE);
      if( rt == TRUE && ( (int)chBuf[0] == 'F' || (int)chBuf[0] == 'M' ||
                     (int)chBuf[2] == 'F' || (int)chBuf[2] == 'M') )
         {
  for( j=0; j< strlen( chBuf); j++)
  DebugDisplayOneChar( chBuf[j]);
         rt = ( (int)chBuf[0] == 'M' || (int)chBuf[2]=='M')?  TRUE: FALSE;
         }
      else
        {
        for( count = 0; count < 30 && bSure == FALSE; count++)
          {
          chBuf[0] = 0;
          chBuf[1] = 0;
          chBuf[2] = 0;
  fprintf( stderr, "%d,", count);
          rt = motCurrAxisSendStringAndGetLine( CMD_REPORT_STATUS, chBuf,
  CH_BUFFER_SIZE);
           if( rt == TRUE && ( (int)chBuf[0] == 'F' || (int)chBuf[0] == 'M' ||
                          (int)chBuf[2] == 'F' || (int)chBuf[2] == 'M'   ) )
              bSure = TRUE;
          }
        rt = ( (int)chBuf[0] == 'M' || (int)chBuf[2]=='M')?  TRUE: FALSE;
        }
      }
    return( rt);
    }
```

```
/*****************************************************
 * switch to specified axis
 *****************************************************
 */
extern BOOL mot_SwitchAxis( MOT_AXIS Axis)
  {
  BOOL      rt = FALSE;

  rt = motWakeupAxisAndCmd( Axis, NULL);
  return rt;
  }

#define WAKEUNIT_BUF_SIZE 70
extern BOOL mot_SendWakeUnitString( MOT_AXIS Axis)
  {
  BOOL      rt = FALSE;
  char chBuf[WAKEUNIT_BUF_SIZE + 2];
  char      *chPtr;

  CurrAxis = Axis;
  switch( Axis)
    {
    case  eAxisX: CurrSGIPort = SGIPortX;
                  CurrhdStage = hdStageX;
                  chPtr = CMD_WAKE_X;
                  break;
    case  eAxisY: CurrSGIPort = SGIPortY;
                  CurrhdStage = hdStageY;
                  chPtr = CMD_WAKE_Y;
                  break;
    case  eAxisZ: CurrSGIPort = SGIPortZ;
                  CurrhdStage = hdStageZ;
                  chPtr = CMD_WAKE_Z;
                  break;
    }
  if( chPtr)
    rt = rs2_FlushQueSendString( CurrhdStage, CurrSGIPort, chPtr);
  if( rt == TRUE)
    rt = rs2_GetOneLine(CurrhdStage, CurrSGIPort, chBuf, WAKEUNIT_BUF_SIZE, 2);
  if( rt == TRUE)
    {
    fprintf( stderr, "str=%s\n", chBuf);
    rt = rs2_FlushQueSendString( CurrhdStage, CurrSGIPort, CMD_SHORT_CMD);
    }
  return rt;
  }
/***********************************************************
 * move motor to specified position, on currently selected
 * axis
 ***********************************************************
 */
extern BOOL mot_CurrAxisAbsMove( int TargetPos)
  {
  BOOL      rt = FALSE;
  char chBuf[30];

  sprintf( chBuf, CMD_ABS_MOVE, TargetPos);
  rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf);
  return rt;
  }
/**********************************************
 * get one char from current axis
 **********************************************
 */
extern BOOL mot_CurrAxisGetOneChar( BYTE * ch)
  {
  BOOL  rt;
  rt = rs2_GetOneChar( CurrhdStage, CurrSGIPort, ch, TIMEOUT_CHAR);
  return( rt );
  }


/*****************************************************************************
 * send command to wakeup the axis
 * and if command is specified, send it also
 ********************************************************************************
 */
static BOOL motWakeupAxisAndCmd(MOT_AXIS Axis, char *pCmd)
  {
  BOOL       rt = FALSE;
  char                  *pWake= NULL;

  if( (Axis == eAxisX && bXOnline == TRUE) ||
     (Axis == eAxisY && bYOnline == TRUE) || (Axis == eAxisZ && bZOnline ==
TRUE) )
    {
    CurrAxis = Axis;
    switch( Axis)
      {
      case eAxisX: CurrhdStage = hdStageX;
                   CurrSGIPort = SGIPortX;
                   pWake = CMD_WAKE_X;
                   break;
      case eAxisY: CurrhdStage = hdStageY;
                   CurrSGIPort = SGIPortY;
                   pWake = CMD_WAKE_Y;
                   break;
      case eAxisZ: CurrhdStage = hdStageZ;
                   CurrSGIPort = SGIPortZ;
                   pWake = CMD_WAKE_Z;
                   break;
      }
    if( hdStageZ != hdStageX)
      rt = TRUE;
    else
      {
      if( pWake)
        rt = motNoCheckWakeUnit( CurrhdStage, CurrSGIPort, pWake);
      }
    if( rt == TRUE && pCmd)
      rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, pCmd);
```

```
        }
  if( rt == FALSE)
  fprintf( stderr, "wake%d=%d,", Axis, rt);
    return rt;
    }


  /**************************************************************************
   * send command to wakeup the axis
   **************************************************************************
   */
  static BOOL motWakeupAxisFast(MOT_AXIS Axis)
    {
    BOOL      rt = FALSE;
    char      *pWake;

    if( (Axis == eAxisX && bXOnline == TRUE) ||
       (Axis == eAxisY && bYOnline == TRUE) || (Axis == eAxisZ && bZOnline ==
  TRUE) )
      {
      CurrAxis = Axis;
      switch( Axis)
        {
        case  eAxisX:  CurrhdStage = hdStageX;
                       CurrSGIPort = SGIPortX;
                       pWake = CMD_WAKE_X;
                       break;
        case  eAxisY:  CurrhdStage = hdStageY;
                       CurrSGIPort = SGIPortY;
                       pWake = CMD_WAKE_Y;
                       break;
        case  eAxisZ:  CurrhdStage = hdStageZ;
                       CurrSGIPort = SGIPortZ;
                       pWake = CMD_WAKE_Z;
                       break;
        }
      if( hdStageZ != hdStageX)
```

```
            rt = TRUE;
          else
            {
            if( pWake)
              rt = wake_unit_funct_fast( CurrhdStage, CurrSGIPort, pWake);
            }
          }
      if( rt == FALSE)
      fprintf( stderr, "wake%d=%d,", Axis, rt);

        return rt;
        }
      /*************************************************************************
       * send a command to neat and wait for a *
       *************************************************************************
       */
      static BOOL motSendAndWaitStar( HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
      *pCmd)
        {
        BOOL      rt = FALSE;

        if( hdStageZ != hdStageX)
          {
          if( rs2_FlushQueSendString(hdStage, SGIPort, pCmd) == TRUE)
            rt = rs2_WaitForResp(hdStage, SGIPort, '*', TIMEOUT_CHAR);
          }
        else
          {
          if( rs2_SendStringWithPerCharHandShake(hdStage, SGIPort, pCmd) == eRS2StatusOK)
            rt = rs2_WaitForResp(hdStage, SGIPort, '*', TIMEOUT_CHAR);
          }
        return rt;
        }


      /*************************************************************************
       * This function is copied from NEAT    3/25/93
```

```
 *
 * wake_unit_funct:  wakes 310 unit given axis index, returns TRUE if OK, *
 *           FALSE if bad axis index value or I/O error          *
 *************************************************************************
 */
static BOOL motNoCheckWakeUnit( HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*WakeCmd)
  {
  int         count,dval;
  BOOL            rt = FALSE;

  command_noecho(hdStage, SGIPort, "");       /* send out CR to clear and delay */
  command_noecho(hdStage, SGIPort, CMD_WAKE_NOBODY);
  command_noecho(hdStage, SGIPort, "");       /* send out CR to clear and delay */

  command_noecho(hdStage, SGIPort, WakeCmd);      /* send out command, don't expect echo
*/
  if(command_noresp(hdStage, SGIPort, "") == TRUE)       /* expect echo */
    rt = TRUE;
  else
    {     /* unit not responding--retry */
    for(count=0; count<5 && rt == FALSE; count++)
      {
      command_noecho(hdStage, SGIPort, WakeCmd);     /* resend wake command--don't
check echo */
      if(command_noresp(hdStage, SGIPort, WakeCmd) == TRUE  ) /* resend again checking
echo */
        rt = TRUE;
      }
if( count >1)
fprintf( stderr, "WakeCt=%d,", count);
    }
  return rt;
  }

/*********************************
 * simplied wakeup, good for
 * getting XY stage position
 ********************************
 */
static BOOL wake_unit_funct_fast(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*WakeCmd)
  {
  int           count,dval;
  BOOL              rt = FALSE;

  for(count=0; count<5 && rt == FALSE; count++)
    {
    command_noecho(hdStage, SGIPort, WakeCmd);          /* send out command, don't
expect echo */
    if(command_noresp(hdStage, SGIPort, WakeCmd) == TRUE  ) /* resend again checking
echo */
      rt = TRUE;
    }

if( count >1)
fprintf( stderr, "WakeCt=%d,", count);

  return rt;
  }



/************************************************************
 ************************************************************
 */
#define NO_ECHO_LOOP_CT              5
#define NO_ECHO_WAIT_MS              5

static BOOL command_noecho(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*cmdstr)
  {
  BOOL      rt = FALSE;
  BYTE             ch;
```

```
    int            TimeoutCt;

    rt = rs2_FlushQueSendString( hdStage, SGIPort, cmdstr);
    if( rt == TRUE)
      {
      TimeoutCt = NO_ECHO_LOOP_CT;
      do
        {
        if(rs2_NonblockGetOneChar( hdStage, SGIPort, &ch) == TRUE)
          TimeoutCt = NO_ECHO_LOOP_CT;
        util_WaitMs( NO_ECHO_WAIT_MS);
        }while(--TimeoutCt > 0);
      }
    return( rt );
    }

/**************************************************************************
 * This function is for NEAT compatiblity
 **************************************************************************
 */

static BOOL command_noresp(HANDLE hdStage, RS232_PORT_SELECT SGIPort, char
*cmdstr)
   {
   BOOL      rt;

   rt = motSendAndWaitStar( hdStage, SGIPort, cmdstr);
   return rt;
   }
/**************************************************************************
 * THis function is here for NEAT compatibility
 * send_echostr:  sends given string over serial port, corfirming an echo *
 *                on each character sent, and appends a CR onto the       *
 *                string and looks for no more characters, returns TRUE   *
 *                if no I/O errors and all characters in string were      *
 *                echoed, returns FALSE if I/O error or bad or no echo,   *
 *                stops sending characters after the 1st bad one          *
 ***************************************************************************
 */
static RS2_CMD_STATUS   send_echostr(HANDLE hdStage, RS232_PORT_SELECT SGIPort,
char *str)
  {
  RS2_CMD_STATUS rtStat ;
  rtStat = rs2_SendStringWithPerCharHandShake( hdStage, SGIPort, str);
  return rtStat;
  }



/*******************************************************
 * testing gmi board:
 * PORT a and B need to looped back
 * LaserPwrControl node must be present
 *
 *******************************************************
 */
#define RANDOM_SIZE     500
#define READ_LOOP       (RANDOM_SIZE / 20)
#define         PORTA           2
#define PORTB           3

extern BOOL mot_UltrapGMITest( int iLoops)
  {
  BYTE            *InputA , *OutputA, *InputB, *OutputB;
  int         iCt, IAIndex, IBIndex, Vers;
  int         iChkCt, iRdCt, iChCt;
  BOOL            rt = FALSE, bHold, bOK;

#ifdef DO_GMI_TEST

  InputA  = UvMalloc( RANDOM_SIZE +10 );
  InputB  = UvMalloc( RANDOM_SIZE +10 );
  OutputA = UvMalloc( RANDOM_SIZE +10 );
  OutputB = UvMalloc( RANDOM_SIZE +10 );
```

```
    fprintf( stderr, "\nGMI rs232 ports need to be conected to each other.\n");
    fprintf( stderr, "Laser power control node must be present.\n");
    fprintf( stderr, "This test sends random char between the 2 port and read sw\n");
    fprintf( stderr, "version number from laser power control node.\n");
    fprintf( stderr, "Will do %d times.\n", iLoops );

    for( iCt= 0; iCt < RANDOM_SIZE; iCt++)
      {
      OutputA [iCt] = rand() & 0xff;
      OutputB [iCt] = rand() & 0xff;
      }
    if( rs2_initialize( hdStage, PORTA, BAUDRATE96) == hdStage )
      if( rs2_initialize( hdStage, PORTB, BAUDRATE96) == hdStage)
        {
        bOK = TRUE;
        rt  = TRUE;
        for( iCt = 0; iCt < iLoops && rt == TRUE; iCt++)
          {
fprintf( stderr, "%d,", iCt);
          lAIndex = 0;
          lBIndex = 0;                  /* swap */
          memcpy( InputA,  OutputA, RANDOM_SIZE);
          memcpy( OutputA, OutputB, RANDOM_SIZE);
          memcpy( OutputB,  InputA, RANDOM_SIZE);

          rs2_SendOneLine( hdStage, eGMI_RS2_0, (char *)OutputA, RANDOM_SIZE);
          rs2_SendOneLine( hdStage, eGMI_RS2_1, (char *)OutputB, RANDOM_SIZE);
          for( iRdCt = 0; iRdCt < READ_LOOP  &&
                        ( lAIndex < RANDOM_SIZE || lBIndex < RANDOM_SIZE ) &&
                        bOK == TRUE;          iRdCt++ )
            {
            for( iChCt = 0; iChCt< 50 &&  lAIndex < RANDOM_SIZE; iChCt++)
              {
              bHold = rs2_NonblockGetOneChar( hdStage, eGMI_RS2_0, &InputA[lAIndex]);
              if( bHold == TRUE)
                lAIndex++;
```

```
                  }
                for( iChCt=0; iChCt < 50 && IBIndex < RANDOM_SIZE; iChCt++)
                  {
                  bHold = rs2_NonblockGetOneChar( hdStage, eGMI_RS2_1, &InputB[IBIndex]);
                  if( bHold == TRUE)
                    IBIndex++;
                  }
                bOK = lonimg_QueryNetVar( LASER_PWR_NODE_ADDR,
    LASER_SW_REV_NV_ADDR, &Vers, TRUE);
                }
            , if( iRdCt >= READ_LOOP)
                fprintf( stderr, "timeout, no responds rx: A=%d,B=%d,\n", IAIndex, IBIndex );

              for( iChkCt= 0; iChkCt < RANDOM_SIZE && rt == TRUE; iChkCt++)
                {
                if( (int)InputA[iChkCt] != (int)OutputB[iChkCt] )
                  {
                  fprintf( stderr, "rs232 error,rx from PORTA at %d,", iChkCt);
                  rt = FALSE;
                  }
                if( (int)InputB[iChkCt] != (int)OutputA[iChkCt] )
                  {
                  fprintf( stderr, "rs232 error,rx from PORTB at %d,", iChkCt);
                  rt = FALSE;
                  }
                }
              fprintf( stderr, "firstchar=%X,%X,%X,%X,\n", InputA[0], OutputB[0],
                                                InputB[0], OutputA[0] );
              if( bOK != TRUE)
                {
                fprintf( stderr, "lon_error,");
                rt = FALSE;
                }
              }
            }

    UvFree( InputA);
    UvFree( InputB);
    UvFree( OutputA);
    UvFree( OutputB);
#else
fprintf( stderr, "gmi test not implemented for now. \n");
#endif

    return rt;
    }
/*****************************************************
 * set current stagelimit to specified counts
 *****************************************************
 */
extern BOOL mot_SetCurrentStgPositiveLim( int StgCt)
    {
    char chBuf[80];
    BOOL      rt;

    sprintf( chBuf, CMD_POSITIVE_LIM,  StgCt);
    rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf );
    return rt;
    }
extern BOOL mot_SetStgPositiveLim(MOT_AXIS Axis, int iCt)
    {
    char chBuf[80];
    BOOL      rt;

    sprintf( chBuf, CMD_POSITIVE_LIM,  iCt);
    rt = motWakeupAxisAndCmd( Axis, chBuf);
    return rt;
    }

extern BOOL mot_SetCurrentStgNegativeLim( int StgCt)
    {
    char chBuf[80];
    BOOL      rt;
```

```
  sprintf( chBuf, CMD_NEGATIVE_LIM,  StgCt);
  rt = motSendAndWaitStar(CurrhdStage, CurrSGIPort, chBuf );
  return rt;
  }
extern BOOL mot_SetCurrentStgJoyStickGain( int iJGain)
  {
  char chBuf[80];
  BOOL      rt;

  sprintf( chBuf, CMD_JOYSTICK_GAIN,  iJGain);
  rt = motSendAndWaitStar( CurrhdStage, CurrSGIPort, chBuf );
  return rt;
  }
extern BOOL mot_SetStgJoyStickGain( MOT_AXIS Axis, int iJGain)
  {
  BOOL      rt = FALSE;

  if( motWakeupAxisAndCmd( Axis, NULL) == TRUE)
    rt = mot_SetCurrentStgJoyStickGain( iJGain);
  return rt;
  }
/*************************************************
 * debug
 *************************************************
 */
static void DebugDisplayOneChar( BYTE  chBuf)
  {
  if( ( (int)chBuf > ' ')  && ((int)chBuf < 0x7f) )
    fprintf( stderr, "%c", chBuf);
  else
    fprintf( stderr, "<%x>", chBuf);
  }
```

☐

RECTIFIED SHEET (RULE 91)

rs232.c

```
/******************************************
 * Ultrapointe Corp. copyright 1992,93,94
 *
 * Interface to  rs232
 * for info: see man page serial/termio:
 *  man serial
 *  man termio
 * and page 75-89 of Advance UNIX programming by Marc J. Rochkind
 *
 * Note:
 * 4 ports supported in UV:
 * PortNum = 0,1,2,3  = SGI_PORT1, SGI_PORT2, ULTRA_PORT0, ULTRA_PORT1
 * port assignment:
 * stage: PortNum 0,1,2
 * robot: PortNum 0,1,3
 *
 * SGI_PORT1 and SGI_PORT2 may be used as terminal by the
 * system. Make sure serial port used here is turned OFF by the
 * system, and uv has r/w access to /dev/ttyd1/2
 *
 ******************************************
 */

/*
 #define MOTOR_DEBUG
*/
#ifdef Irix5plus
#define NeedFunctionPrototypes 1
#else
#define FUNCPROTO
#endif

#include <stdio.h>
#include <termio.h>
#include <errno.h>
#include <setjmp.h>
#include <signal.h>
```

RECTIFIED SHEET (RULE 91)

```
#include <unistd.h>
#include <limits.h>             /* sginap */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <X11/Intrinsic.h>

#include "uv_util.h"
#include "rs232.h"
#include "uldd.h"

/* cannot find the prototype include file for ioctl..... workaround and defined here....
 */
extern int ioctl (int fildes, int request, ...);


#define MAX_LINE 100

#define PORT1 "/dev/ttyd1"
#define PORT2 "/dev/ttyd2"
#define PORT1X "/dev/ttyd05%d"

#define TIMEOUT_CHAR          1    /* something that should have no delay */
#define TIMEOUT_LINE          10   /* something that is slow */
#define TIMEOUT_RESPONSE 10

#define NEAT_INTER_CHAR_WAIT     1          /* wait 1ms per char */

static jmp_buf env_alarm;

static BOOL rsSetRaw(HANDLE hdRS232, int BaudRate);
static  void sig_alarm( int signo);
static void DebugDisplayOneChar( BYTE  chBuf);
static BOOL rs2SendOneChar( HANDLE hdRS232, RS232_PORT_SELECT SGIPort, char ch);


/****************************************************
```

```
 * initialization:
 *   PortNum:  0= PORT1,1=PORT2
 *   BaudRate: 0= 9600, 1=19200, 2=38400
 * see: man page on termio
 *****************************************************
 */
extern int rs2_initialize( int hUldd, int PortNum, int BaudRate )
  {
  char   *pPort, chBuf[80];
  int  hdRS232 = -1;
  CMD_RS2_CONTROL RstCtl;

  switch (PortNum)
      {
      case 0:
      case 1:
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
      case 16:
      case 17:
         if( PortNum >= 10 && PortNum <= 17 )
           {
           sprintf( chBuf, PORT1X, PortNum -10);
           pPort = chBuf;
           }
         else
           pPort = ( PortNum == 1)? PORT2 :PORT1;

         hdRS232 = open( pPort, O_RDWR | O_NOCTTY);
         if( hdRS232 >= 0)
           {
           if( rsSetRaw( hdRS232, BaudRate) == FALSE)
             {
                close( hdRS232);
                hdRS232 = -1;
                }
            }
          break;
        case 2:
        case 3:
          RstCtl.CtlCode = ( PortNum == 2)? eCmdRS2InitPort0 : eCmdRS2InitPort1;
          switch( BaudRate)
            {
            case BAUDRATE192: RstCtl.BaudRate = GMI_BAUD_19200; break;
            case BAUDRATE384: RstCtl.BaudRate = GMI_BAUD_38400; break;
            case BAUDRATE96:
            default:          RstCtl.BaudRate = GMI_BAUD_9600; break;
            }

         if( ioctl( hUldd, eCmdRS2Control, &RstCtl ) >= 0)
              {
              hdRS232 = hUldd;
              }
else
fprintf( stderr, "ioctlerr=%X, ", errno );


         break;
      }
  if( hdRS232 < 0)
      fprintf( stderr, "cannot open, error=%d,", errno);
  return hdRS232;
  }


extern BOOL rs2_ClosePort( HANDLE hdRS232, RS232_PORT_SELECT SGIPort)
      {
      if(SGIPort == eSGI_RS2_Port && hdRS232 >=0)
         close(hdRS232);
      return TRUE;
      }
```

```
/****************************************************************
 * send one string to rs232.. For the NEAT controller
 * check for echo of each char
 * up to 2 different char recieved per one sent are tolerated
 * NEAT's reponse may have 'F' mixed in the char stream
 ****************************************************************
 */
#define NUM_DIFF_TOLERATED        3
extern  RS2_CMD_STATUS rs2_SendStringWithPerCharHandShake( HANDLE hdRS232,
RS232_PORT_SELECT SGIPort, char * chPtr)
  {
  RS2_CMD_STATUS rtStat=  eRS2StatusBad;
  BYTE  chTerm = '\r', chBuf;
  int          iDiffCt;
  BOOL              rt;

  if( hdRS232 > =0 && strlen(chPtr) > 0 )
    {
    rs2_FlushInputQueue(hdRS232, SGIPort);
    rtStat = eRS2StatusOK;
    do
      {
      rs2SendOneChar( hdRS232, SGIPort, *chPtr);
      rtStat = eRS2StatusBad;
      for( iDiffCt = 0; iDiffCt < NUM_DIFF_TOLERATED && rtStat == eRS2StatusBad;
iDiffCt++)
        {
        rt = rs2_GetOneChar( hdRS232, SGIPort, &chBuf,TIMEOUT_CHAR);
        if ( rt == FALSE)
           rtStat = eRS2StatusTimeout;
        else
           {
           if( ((int)chBuf & 0x5f) == ((int)*chPtr & 0x5f) )
             rtStat = eRS2StatusOK;
           else
             {
fprintf( stderr, "chrx=%X,", chBuf);
```

```
                    util_WaitMs( NEAT_INTER_CHAR_WAIT);
                    }
                  }
                }
            chPtr++;
            } while( rtStat == eRS2StatusOK && (int)*chPtr != '\r' && (int)*chPtr != '\0') ;
         rs2SendOneChar( hdRS232, SGIPort, chTerm) ;
         }
       return ( rtStat);
       }
/*************************************************************
 * SGIPort = eSGI_RS2_Port:  from  standard indigo port
 *         = eGMI_RS2_0/1: GMI board, hdRS232=0: channel A
 *************************************************************
 */
static BOOL rs2SendOneChar( HANDLE hdRS232, RS232_PORT_SELECT SGIPort, char ch)
  {
  CMD_RS2_WR_PACKET WrPack;
  int   iWr= 0;
  BOOL      rt = FALSE;

  if( SGIPort == eSGI_RS2_Port )
    iWr = write( hdRS232, &ch, 1 ) ;
  else
    {
    WrPack.Line     = (SGIPort == eGMI_RS2_0)? 0: 1;
    WrPack.iMaxChar      = 1;
    WrPack.iWr     = 0;
    WrPack.chBuf   = &ch;
    if( ioctl( hdRS232, eCmdRS2BlockWrite, &WrPack ) >= 0)
       iWr = WrPack.iWr;
    }
  return (rt = ( iWr> 0)? TRUE: FALSE );
  }
/*************************************************************
 * bSGIPort = TRUE:  from  standard indigo port
 *          = FLASE: GMI board, hdRS232=0: channel A
 *************************************************************
 */
extern int rs2_SendOneLine( HANDLE hdRS232. RS232_PORT_SELECT SGIPort. char *chPtr,
int NumChar)
  {
  CMD_RS2_WR_PACKET WrPack;
  int   iWr= 0;

  if( SGIPort == eSGI_RS2_Port)
    iWr = write( hdRS232. chPtr. NumChar ) ;
  else
    {
    WrPack.Line    = (SGIPort == eGMI_RS2_0)? 0: 1;
    WrPack.iMaxChar      = NumChar;
    WrPack.iWr    = 0;
    WrPack.chBuf   = chPtr;
    if( ioctl( hdRS232. eCmdRS2BlockWrite. &WrPack ) >= 0)
       iWr = WrPack.iWr;
    }
  return iWr;
  }

/*********************************************
 * send a string to rs232. dont wait for response
 *********************************************
 */
extern BOOL rs2_FlushQueSendString( HANDLE hdRS232. RS232_PORT_SELECT SGIPort.
char * chPtr)
  {
  BOOL     rt = FALSE;

  if( hdRS232 >=0 && strlen(chPtr) > 0 )
    {
    rs2_FlushInputQueue( hdRS232. SGIPort);
    if( rs2_SendOneLine( hdRS232. SGIPort. chPtr. strlen( chPtr) ) > 0)
       rt = TRUE;
    }
  return rt;
  }
/********************************************
 * send a string to rs232, no flush, no wait for response
 ********************************************
 */
extern BOOL rs2_SendStringNoWaitNoFlush( HANDLE hdRS232, RS232_PORT_SELECT
SGIPort, char * chPtr)
  {
  BOOL      rt = FALSE;

  if( hdRS232 > =0 && strlen(chPtr) > 0 )
    {
    if( rs2_SendOneLine( hdRS232, SGIPort, chPtr, strlen( chPtr) ) > 0)
      rt = TRUE;
    }
  return rt;
  }
/************************************************
 * wait for the correct respond
 * return TRUE if received correct responds
 *       FALSE if timed out
 ************************************************
 */
extern BOOL rs2_WaitForResp(HANDLE hdRS232, RS232_PORT_SELECT SGIPort, char ch,
int Seconds)
  {
  BYTE chBuf[10];
  BOOL      rt= FALSE;

  if( hdRS232 > =0)
    {
    do
      {
      rt = rs2_GetOneChar( hdRS232, SGIPort, chBuf, Seconds);
      } while ( rt == TRUE && (int)chBuf[0] != (int)ch );
    }
  return rt;
  }
/*********************************************************
 * wait to get one char
 * return 1 if got a char,
 *       0 if not: file closed, or timeout
 *********************************************************
 */
extern BOOL rs2_GetOneChar( HANDLE hdRS232, RS232_PORT_SELECT SGIPort, BYTE
*ch, int Seconds)
  {
  BOOL rt = FALSE;
  CMD_RS2_RD_PACKET RdPack;

  if( hdRS232 >=0)
    {
    if( SGIPort == eSGI_RS2_Port)
      {
      signal( SIGALRM, sig_alarm);
      if( setjmp( env_alarm) != 0)
         return FALSE;                 /* return 0 if timeout */
      else
         {
         alarm( Seconds);
         if(  read( hdRS232, ch, 1) > 0)
            rt = TRUE;
         alarm( 0);
         }
      }
    else
      {
      RdPack.Line        = ( SGIPort == eGMI_RS2_0)? 0: 1;
      RdPack.iMaxChar = 1;
      RdPack.iRd     = 0;
      RdPack.chBuf       = (char *)ch;
      RdPack.iTimeout = Seconds * CLK_TCK;
      if( ioctl( hdRS232, eCmdRS2BlockRead, &RdPack ) >= 0)
```

```
            {
            if( RdPack.iRd == 1)
              rt = TRUE;
            }
          }
        }
      return rt;
      }

    /**********************************************************
     * get one char if available, assume ttyport ( if used) set to non-block
     * return TRUE   if got a char,
     *        FALSE  if not: empty, file closed, or timeout
     **********************************************************
     */
    extern BOOL rs2_NonblockGetOneChar( HANDLE hdRS232, RS232_PORT_SELECT SGIPort,
    BYTE *ch)
      {
      BOOL rt = FALSE;
      CMD_RS2_RD_PACKET RdPack;

      if( hdRS232 >=0)
        {
        if( SGIPort == eSGI_RS2_Port)
          {
          if( rs2_SetToNonBlockedRead(  hdRS232) == TRUE)
            {
            if ( read( hdRS232, ch, 1) > 0)
              rt = TRUE;
            }
          rs2_SetToBlockedRead( hdRS232);
          }
        else
          {
          RdPack.Line          = ( SGIPort == eGMI_RS2_0)? 0: 1;
          RdPack.iMaxChar = 1;
          RdPack.iRd     = 0;
```

```
            RdPack.chBuf          = (char *)ch;
            if( ioctl( hdRS232, eCmdRS2NonBlockRd, &RdPack ) >= 0)
              {
              if( RdPack.iRd > 0)
                rt = TRUE;
              }
            }
          }
        return rt;
        }




      /********************************************************
       * software interrupt
       * from alarm timer
       ********************************************************
       */
      static  void sig_alarm( int signo)
        {
        longjmp( env_alarm, 1);
        }

      extern BOOL rs2_FlushInputQueue( HANDLE hdRS232, RS232_PORT_SELECT SGIPort)
        {
        BOOL      rt = FALSE;
        char chBuf[MAX_LINE];
        CMD_RS2_RD_PACKET RdPack;

        if( SGIPort == eSGI_RS2_Port)
          {
          if( rs2_SetToNonBlockedRead(  hdRS232) == TRUE)
            {
            while( read( hdRS232, chBuf, 1) > 0)
               DebugDisplayOneChar( chBuf[0]);
        rt = rs2_SetToBlockedRead( hdRS232);
        }
      }
    else
      {
      rt = TRUE;
      do
        {
        RdPack.Line         = (SGIPort == eGMI_RS2_0)? 0: 1;
        RdPack.iMaxChar = MAX_LINE;
        RdPack.iRd   = 0;
        RdPack.chBuf        = chBuf;
        if( ioctl( hdRS232, eCmdRS2NonBlockRd, &RdPack ) < 0)
          rt = FALSE;
        } while ( rt == TRUE && RdPack.iRd > 0);
      }
    return rt;
    }
/*********************************************************
 * get one line, terminated by LF or CR
 * return FALSE if TIMEOUT, or max line length exceeded
 *********************************************************
 */
extern BOOL rs2_GetOneLine( HANDLE hdRS232, RS232_PORT_SELECT SGIPort, char
*chBuf, int iMax,
                              int Seconds)
  {
  int  iCt;
  BOOL     rt = FALSE, bEnd = FALSE;

  if( hdRS232 >=0 && iMax > 0)
    {
    rt = TRUE;
    for( iCt =0; iCt < iMax && bEnd == FALSE && rt == TRUE; iCt++)
      {
      rt = rs2_GetOneChar( hdRS232, SGIPort, (BYTE *)&chBuf[iCt], Seconds);
      if( rt == TRUE)
```

```
              {
              DebugDisplayOneChar( chBuf[iCt]);
              if( (int)chBuf[iCt] == 0xd || (int)chBuf[iCt] == 0xa)
                {
                bEnd = TRUE;
                chBuf[iCt] = '\0';
                }
              }
            }
          }
     if( rt == FALSE )
       chBuf[0] = '\0';
     return( rt);
     }
/************************************************
 * set to raw mode
 * BaudRate: 0= 9600;
 *           1= 19200
 ************************************************
 */
static BOOL rsSetRaw(HANDLE hdRS232, int BaudRate)
    {
    struct termio tbuf;
    BOOL      rt= FALSE;
    char ibuf[MAX_LINE];

    if( hdRS232 >=0)
        {
        if( ioctl( hdRS232, TCGETA, &tbuf) == -1)
          fprintf(stderr, "ioctl get:error ");
       else
          {
          tbuf.c_iflag &= ~(INLCR | ICRNL | IUCLC | ISTRIP | IXON | BRKINT);
          tbuf.c_oflag &= ~OPOST;
          tbuf.c_lflag &= ~(ICANON | ISIG | ECHO | XCASE);
          tbuf.c_cflag &= ~(CBAUD | CSIZE | PARENB ); /* baud, 8bit parity */
          switch ( BaudRate )
```

RECTIFIED SHEET (RULE 91)

```
                {
                case BAUDRATE192:
                  tbuf.c_cflag |= ( B19200 | CS8 | CREAD | CLOCAL);
                   break;
                case BAUDRATE384:
                  tbuf.c_cflag |= ( B38400 | CS8 | CREAD | CLOCAL);
                   break;
                case BAUDRATE96:
                default:
                  tbuf.c_cflag |= ( B9600 | CS8 | CREAD | CLOCAL);
                   break;
                }
            tbuf.c_cc[0] = 0xff;
            tbuf.c_cc[1] = 0xff;
            tbuf.c_cc[2] = 0xff;
            tbuf.c_cc[3] = 0xff;
            tbuf.c_cc[VMIN] = 1;    /* min */
            tbuf.c_cc[VTIME] = 1;   /* time */
            if( ioctl(hdRS232, TCSETAF, &tbuf) == -1)
               fprintf(stderr, "ioctl set: error ");
            else
              {
              rt = rs2_FlushInputQueue( hdRS232, eSGI_RS2_Port) ;
              }
            }
      }
   return rt;
   }
/****************************************************
 * set to BLOCKed read
 * see man page termio, and fcntl
 ****************************************************
 */
extern BOOL rs2_SetToBlockedRead( HANDLE Receiver)
   {
   int  FFlag;
   FFlag = fcntl( Receiver, F_GETFL, 0);
```

```
  if( fcntl( Receiver, F_SETFL, FFlag &( ~FNDELAY) ) == 0)
    return TRUE;
  else
    return FALSE;
  }
/**************************************************
 * set to non-BLOCKed read
 * see man page termio, and fcntl
 **************************************************
 */
extern BOOL rs2_SetToNonBlockedRead( HANDLE Receiver )
  {
  int  FFlag;
  FFlag = fcntl( Receiver, F_GETFL, 0);
  if( fcntl( Receiver, F_SETFL, FFlag | FNDELAY) == 0)
    return TRUE;
  else
    return FALSE;
  }
/**********************************************
 * debug
 **********************************************
 */
static void DebugDisplayOneChar( BYTE  chBuf)
  {
#ifdef MOTOR_DEBUG
  if( chBuf > ' ')
    fprintf( stderr, "%c", chBuf);
  else
    fprintf( stderr, "<%x>", chBuf);
#endif
  }
```

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX F

M-2464-1P US

1/6/95

```
/*****************************************************
 * (c) Copyright 1994
 * Ultrapointe Corp.
 *
 * page scan fine Z autofocus
 * host computer support functions
 *
 * functions used but not listed:
 *
 * lon_APICmd()
 *       set the specified subsystem parameter to specified value
 *
 * lon_APIQuery()
 *       query the status of the the specified subsystem parameter
 *
 * cfg_GetLockRdConfig
 *       Get pointer to system configuration data structure
 *
 *****************************************************
 */
        /* system hardware image */
static UI_LON_IMAGE   UIImage;

/*****************************************************************
 * controlling the page scanning hardware
 *
 *
 *
 *                                         *                                       *
 *                                   *    *                                 *
 *                              *            *                          *
 *                     *                   *                     *
 *    *          *                  *            *
 *     *     *                                  *     *
 *      *                             *
 *       | 1 |               2              | 3     | 1      |      2       regions
```

1/6/95

```
*
* region 2 gives the visible data
*
* scanner f= 7.875 kHz, duty cycle for y=256 is 92%, y=512 is 96%
*
* example: y=256  (region 2)  =              256 cycles
*          retrace( region 3+1)= 5ms      = 39.4 cycles
*
*
*                           region(1+2)              region(1+2)
* for a 92% duty cycle = ------------     = .92  =   ----------     -> region3 = 23.6
*                          region 1+2+3               295.4     then, region1 =39.4-
*                                                                        23.6=15.7
* converting these into RAMPCLK, 8times
*ie: y=256:    VSYNC   = 15.7 * 8 = 125.6 =  126
*                TRACE   =  256 * 8 =         2048
*                RETRACE = 23.6 * 8 = 188.8 =  189
* peak-to-peak-ampl = Incr *( VSYNC + TRACE + 1) = Decr *( RETRACE +2)
* for peak-to-peak = about 12288 -> 6 * ( 126 + 2048 + 1) = 13050
* and adjust DECR to fit,
* after adjusting( duty cycle, peak-to-peak)
* for y=128:   VSYNC   =   131                 duty cycle = 90.4 %
*               TRACE   =  1024
*               RETRACE =   202
*               INCR    =    12
*               DECR    =    68
*             Total Ampl = (1024+131+1) *12 = 13872 -> half height 6936-> start =
*                                                                              1256
* for y=256:   VSYNC   =   127                          92%
*               TRACE   =  2048
*               RETRACE =   190
*               INCR    =     6
*               DECR    =    68
*             Total Ampl = (2048+127+1) *6 = 13056 -> half height 6528 ->start= 1664
*
```

RECTIFIED SHEET (RULE 91)

```
1/6/95
 * for y=512:  VSYNC    =   135                    96%
 *              TRACE    =  4096
 *              RETRACE =   182
 *              INCR     =     3
 *              DECR     =    69
 *            Total Ampl = (4096+135+1) *3 = 12696 -> half height 6348 ->start =
 *                                                                       1844
 *******************************************************************
 */

typedef struct PAGE_SCAN_PARAMS_tag
  {
  int     iStartValue;
  int     iIncr;
  int     iDecr;
  int     iVSync;
  int     iTrace;
  int     iRetrace;
  }PAGE_SCAN_PARAMS;

static PAGE_SCAN_PARAMS PageScanParms128  = { 1256, 12, 68, 131, 1024, 202 };
static PAGE_SCAN_PARAMS PageScanParms256  = { 1664,  6, 68, 127, 2048, 190 };
static PAGE_SCAN_PARAMS PageScanParms512  = { 1844,  3, 69, 135, 4096, 182 };
static PAGE_SCAN_PARAMS PageScanSuperAF   = {    0, 32, 32,   9,  246, 254 };

/****************************************************
 * super fine Autofocus
 * with page scan
 ****************************************************
 */
static BOOL IonuiSuperFineAF( UV_CONFIG *pCfg, OPTICS_PARMS *pLaser, int AFCode,
int *PeakInten)
  {
  BOOL                  rt = FALSE;
  int              iZIndex, YAmp, YStart;
```

1/6/95

```
                        /* get laser current zoom factor*/
iZIndex = lonui_GetIndexFrPixZoomEnum( pLaser->PixZoom);
                        /* get YStart value, and YAmp from configuration */
if( pLaser->YDim == 512)
  {
  YStart = pCfg->StageSpec.FineAFPScanYStart512[iZIndex];
  YAmp   = pCfg->StageSpec.FineAFPScanYAmp512  [iZIndex];
  }
else
  {
  YStart = pCfg->StageSpec.FineAFPScanYStart256[iZIndex];
  YAmp   = pCfg->StageSpec.FineAFPScanYAmp256  [iZIndex];
  }

rt = lon_APICmd( eLonAPIPageYAmpl, 0);
if( rt == TRUE)
  rt = lon_APICmd  ( eLonAPIPageControlReg, PAGE_SCAN_CTRL_SCAN_ON  |
                                             PAGE_SCAN_CTRL_STOP_DAC);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageVSync,     PageScanSuperAF.iVSync);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageTraceTime, PageScanSuperAF.iTrace);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageRetrace,   PageScanSuperAF.iRetrace);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageStartAmpl, YStart);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageIncrAmpl,  PageScanSuperAF.iIncr);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageDecrAmpl,  PageScanSuperAF.iDecr);
if( rt == TRUE)
  rt = lon_APICmd( eLonAPIPageYAmpl,     YAmp);
if( rt == TRUE)
  rt = lon_APICmd  ( eLonAPIPageControlReg, PAGE_SCAN_CTRL_SCAN_ON );
```

1/6/95

```
  if( rt == TRUE)
    rt = Ionimg_QueryNetVar( FAST_Z_NODE_ADDR, AFCode, PeakInten, TRUE);

  Ionui_PageScan( pLaser->YDim, pLaser->PixZoom);
  return( rt);
  }

/**************************************************
 * convert from enum to index
 **************************************************
 */
extern int Ionui_GetIndexFrPixZoomEnum( LASER_PIX_ZOOM  ePixZoom)
  {
  int ZoomIndex;

  switch( ePixZoom)
    {
    case ePixZoom12nm:  ZoomIndex = 3;      break;
    case ePixZoom25nm:  ZoomIndex = 2;      break;
    case ePixZoom50nm:  ZoomIndex = 1;      break;
    case ePixZoom100nm:
    default:            ZoomIndex = 0;      break;
    }
  return ZoomIndex;
  }
/*****************************************************************************
 * initialize /setup page scanner
 * page scanner is now alway on
 *****************************************************************************
 */
extern BOOL Ionui_PageScan( int iYDim, LASER_PIX_ZOOM ePixZoom)
  {
  PAGE_SCAN_PARAMS       *PScanParms;
  BOOL                   rt = FALSE;
  int              XAmp, YAmp, YIndex, ZoomIndex, TurPos, PageCtrlReg;
  UV_CONFIG        *pCfg    = cfg_GetLockRdConfig();
```

1/6/95

```
if( lon_APIQuery( eLonAPITurretPosCmd, &TurPos, 0) == TRUE)
   TurPos--;
else
   TurPos = 0;

if( TurPos < 0 || TurPos >= (HDWR_TURRET_HOME_OBJ -1) )
   TurPos = 0;

switch ( iYDim)
   {
   case 256: PScanParms = &PageScanParms256;
                break;
   case 512: PScanParms = &PageScanParms512;
                break;
   default:     PScanParms = NULL;
                break;
   }

ZoomIndex = lonui_GetIndexFrPixZoomEnum( ePixZoom);

if( PScanParms)
   {
   UIImage.iYCt        = iYDim;
   UIImage.XYZoom      = ePixZoom;

   rt = lon_APICmd( eLonAPIPageYAmpl, 0);
   if( rt == TRUE)
       rt = lon_APICmd  ( eLonAPIPageControlReg, PAGE_SCAN_CTRL_SCAN_ON |
                                               PAGE_SCAN_CTRL_STOP_DAC);
   if( rt == TRUE)
       rt = lon_APICmd( eLonAPIPageVSync,  PScanParms->iVSync);
   if( rt == TRUE)
     rt = lon_APICmd( eLonAPIPageTraceTime, PScanParms->iTrace );
   if( rt == TRUE)
     rt = lon_APICmd( eLonAPIPageRetrace,    PScanParms->iRetrace);
```

1/6/95

```
    if( rt == TRUE)
      rt = lon_APICmd( eLonAPIPageStartAmpl, PScanParms->iStartValue);
    if( rt == TRUE)
      rt = lon_APICmd( eLonAPIPageIncrAmpl,  PScanParms->iIncr);
    if( rt == TRUE)
      rt = lon_APICmd( eLonAPIPageDecrAmpl,  PScanParms->iDecr);
    if( rt == TRUE)
      rt = lon_APICmd( eLonAPIPageXAmpl,
                       pCfg->ConfocalSpec.XScanAmpl[TurPos][ZoomIndex] );
    if( rt == TRUE)
      {
        YAmp = pCfg->ConfocalSpec.YScanAmpl[TurPos][ZoomIndex];
        if( iYDim == 256 )
          YAmp = YAmp/2;

        rt = lon_APICmd( eLonAPIPageYAmpl, YAmp);
        }
    rt = lon_APICmd  ( eLonAPIPageControlReg, PAGE_SCAN_CTRL_SCAN_ON );
    }
  return rt;
  }
```

A METHOD AND APPARATUS FOR PERFORMING AN AUTOMATIC FOCUS OPERATION

Christopher R. Fairley
Timothy V. Thompson
Ken K. Lee

APPENDIX G

M-2464-1P US

RECTIFIED SHEET (RULE 91)

Ultrapointe Model 1000
Laser Imaging System
System User's Manual

Copyright 1993, 1994

Ultrapointe Corporation
163 Baypointe Parkway
San Jose, CA 95134

(408) 894-7080

P/N 000621
Rev. 1.3

DRAFT

Revised: 3 June, 1994

Foreword

Revision 1.3 of the UV1000 software incorporates all changes and features provided in various unreleased versions through 1.2T(u), and issued before 18 May, 1994. This manual revision 1.3 documents all those features. In addition, customer preventative maintenance procedures have been incorporated as Appendix E. Finally, while the manual refers to the LIS 1000 hardware, the manual applies also to the model LIS 500. The manual assumes the LIS 500 has been ordered with full 3-D capability, but if this is not the case, then those features not ordered will not be available.

SAFETY

All user interaction with the instrument is via the front panel switches, the mass storage devices in the drive bay, or the keyboard and display console. The user must be familiar with all external system components, per the Safety and Operating sections of this manual. Servicing is to be performed by factory-authorized personnel only.

The Ultrapointe Model 1000 has been designed to minimize the hazards presented to the user during normal operation of the instrument. All servicing is to be performed by factory-authorized personnel only.

Laser Safety

In accordance with the Regulations for the Administration and Enforcement of the Radiation Control for Health and Safety Act of 1968 (applicable to laser products), a Laser Product Report for this instrument has been forwarded to the Center for Devices and Radiological Health (CDRH). This instrument falls under Class I definitions because there is no user access to laser radiation.

CAUTION: Use of controls or adjustments, or performance of procedures other than those specified herein may result in hazardous radiation exposure.

The protective housing of the instrument should be kept intact during normal operation. Internal to the instrument is a Class IIIb argon-ion laser rated at 25 milliwatts with a maximum possible output of 500 milliwatts. Removal of any part of the upper protective housing can expose the user unnecessarily to hazardous laser radiation. Danger labels reproduced below indicate covers that provide access to hazardous laser light exposure.

DANGER - Laser radiation when
open and interlock defeated.
AVOID DIRECT EXPOSURE TO BEAM

DANGER - Laser radiation when
open. AVOID DIRECT EXPOSURE
TO BEAM

Safety, Continued

High Voltage

Internal to the instrument are components using voltages up to 1000 VDC. Access to and servicing of all internal components is by factory-authorized personnel only.

Moving Parts

As part of routine instrument operations, various parts of the machine make movements automatically and when commanded by the user. THe user should take care to keep body parts and clothing away from the robot area. The user should also ensure that the current robot operation is complete before adding or removing a cassette from the cassette locator plate.

Much of the instrument is attached to an internal frame that is mounted to the external frame and protective covers via pneumatic vibration isolators. The "floating" internal frame can therefore move freely with respect to the covers and creates possible pinch points between floating components and the covers or external frame. Externally accessible floating components include the cassette platform, the robot, and the flat finder.

Inside the instrument, are many moving parts which could create possible hazards, including motors, shutters, and an X/Y/Z stage. Access to components inside the machine with system power on is restricted to factory-authorized service personnel.

Moving parts on the outside of the system include the robot handler, pre-aligner, and wafer shutter door, as well as the floating optics platform. The user should take care to keep body parts and clothing clear of all moving parts at all time.

Table of Contents

1. Introduction 307

2. Principles of operation 311

3. Safety 315
    3.1. OPERATING SAFETY: 315
        3.1.1. Laser Safety: 315
        3.1.2. High Voltage: 316
        3.1.3. Moving Parts: 316

4. Software Installation 318

5. System Usage 319
    5.1. Introduction 319
        5.1.1. Basic Controls 319
            5.1.1.1. Mouse 319
            5.1.1.2. Menu Bars 320
            5.1.1.3. Option Menus 320
            5.1.1.4. Buttons 320
            5.1.1.5. Slider Bars 321
            5.1.1.6 Entry Field 322
    5.2. Data files and organization 322
        5.2.1. Volume and Surface 323
        5.2.2. Wafer map File 324
            5.2.2.1 Defect information 324
            5.2.2.2 Wafer information 325
            5.2.2.3 Identification 325
        5.2.3. Recipe 325
            4.2.3.1 Screen layout 325
            4.2.3.2 Laser scan parameters 326
            4.2.3.3 Defect Sort Criterion 326
        5.2.4. Bit map files: 326

RECTIFIED SHEET (RULE 91)

5.2.5. Defect Code Table files: 326
5.3. Top Menu Bar 327
  5.3.1. File: 327
    5.3.1.1. Image: 327
    5.3.1.2. Recipe: 327
    5.3.1.3. Wafer Map: 328
    5.3.1.4. Defect Code: 329
    5.3.1.5. Exit: 330
  5.3.2. Screen: 330
  5.3.3. Setup: 330
    5.3.3.1. LaserScan Parms: 330
    5.3.3.2. Deskew Options 331
    5.3.3.3 Edit Wafer Map: 331
    5.3.3.4. Defect List Sort: 333
    5.3.3.5 Rearranging Wafers 336
  5.3.4. User: 336
    5.3.4.1. Login: 337
  5.3.5. Config: 337
    5.3.5.1. SECS: 337
    5.3.5.2. Printer: 337
    5.3.5.3 Misc. 337
  5.3.6. Maintenance: 340
    5.3.6.1. Manual stage contr 340
    5.3.6.2. Manual robot control: 340
    5.3.6.3. File Transfer: 340
    5.3.6.4. UserAcct: 342
    5.3.6.5. LON Diagnostics: 344
    5.3.6.6. Calibration: 344
  5.3.7. PrtScreen: 344
5.4. Window and Screen 344
  5.4.1. Common window attributes 345

5.4.1.1. Window sizes. 346

5.4.1.2. Swapable to other window sites on-screen. 346

5.4.1.3. Global and local data. 347

5.4.1.4. Inter-window data exchange. . . . 347

5.4.2. Window types 348

5.4.2.1. Cassette Window 348

5.4.2.2. Wafer Window 351

5.4.2.3. Laser Window 353

5.4.2.4. Microscope window 361

5.4.2.5. 2D Image Window 361

5.4.2.6. 3D Surface Image Window 363/2

5.4.2.7. Library Image Window 363/6

5.4.2.8. Text and status window 363/7

5.4.3. Screen 363/13

5.4.3.1. Switching to a different screen 363/13

5.5. User privilege 363/14

5.6. Operator Mode 363/14

5.6.1. System Startup 363/14

5.6.1.1. Laser imaging system Login: . . 363/15

5.6.1.2. Hardware initialization: 363/15

5.6.2. Wafer loading, cassette window 363/16

5.6.3. Deskew, Text Status window 363/16

5.6.4. Defect location, Wafer window 363/17

5.6.5. Image Acquisition, Laser window 363/18

5.6.5.1. White light only. 363/18

5.6.5.2. White light and laser 363/18

5.6.6. Screen Printing 363/20

5.6.7. Image storage 363/20

5.6.8 Defect Classification 363/20

RECTIFIED SHEET (RULE 91)

5.7. Engineering Mode 363/21
  5.7.1. User type and Operator access control . 363/21
  5.7.2. Recipe preparation 363/22
    5.7.2.1. Screen setup and sequencing . 363/22
    5.7.2.2. Laser parameter control 363/23
    5.7.2.3 Sort criteria 363/23
  5.7.3. Manual Robot control 363/23
  5.7.4. Manual stage control 363/24
    5.7.4.1. Enable/Disable joystick, all axes 363/24
    5.7.4.2. Report Position/encoder count . 363/24
    5.7.4.3. Report sample thickness 363/24
    5.7.4.4. XYstage burn-in 363/24
    5.7.4.5. Stop 363/24
    5.7.4.6. XYTo Load/unload 363/25
    5.7.4.7. Move, absolute or relative 363/25
    5.7.4.8. Set speed 363/25
    5.7.4.9. Set acceleration 363/25
    5.7.4.10. Move to Home 363/25
  5.7.5. Hardware diagnostics 363/25
  5.7.6. System Calibration 363/26
    5.7.6.1. Laser scan XY dimension calibration 363/26
  5.7.7 System configuration 363/26
    5.7.7.1 Screen/Laser saver timer interval 363/26
    5.7.7.2 File DeskewDie Offset 363/26
    5.3.5.3.3 Flat angle offset(deg.) 363/27
    5.3.5.3.4 White light mode 363/27
    5.3.5.3.5 Next die position 363/28
  5.7.8. System Startup 363/28

5.7.8.1. Operating system and windowing system 363/28
5.7.8.2. IRIX Computer user autologin and autostart: 363/29
6. Troubleshooting 363/30
6.1. Robot operation 363/30
6.2. Stage Operation 363/30
6.3. Power fail Recovery 363/30
6.3.1. Wafer recovery 363/31
6.3.2. System component reset 363/31
6.4. LON diagnostics dialog box 363/31
6.4.1 Wafer door interlock. 363/31
6.4.2 Critical parameters 363/31
6.5. System Service 363/31
6.5.1. Software configuration 363/31
6.6. System Messages and Response 363/31
6.6.1. Caution/Information 363/32
6.6.2. Error Condition and Recovery 363/32
6.6.2.1 Floppy disk not responding . . 363/32
6.6.2.2 Printer not printing 363/32
7. Applications Information 363/34
7.1. Wavelength Selection 363/34
7.2. Image interpretation 363/34
7.2.1. Surface reflectivity 363/34
7.2.2. Limits of performance 363/35
7.3. Complex surfaces 363/36
7.3.1. Transparent coatings 363/36
7.3.2. Multilayer structures 363/36
7.3.3. High contrast variations 363/36
7.3.4. Vertical Surfaces 363/37

RECTIFIED SHEET (RULE 91)

8. System Maintenance 363/38
  8.1. System software maintenance 363/38
    8.1.1 Setting Date and Time 363/38
    8.1.2. System shutdown 363/38
  8.2. Maintenance and Laser Safety 363/38
  8.3. Customer Preventative Maintenance Procedures 363/39
  8.4. Extraordinary maintenance 363/39
    8.4.1. Wafer removal 363/39
    8.4.2. Other 363/39

9. Appendices 363/40

  Appendix A 363/40
  A.1 To Install the Indigo System Software 363/40
  A.2 To Install the Ultrapointe Application Software . 363/41
  A.3 InstallApplic Script 363/42
  A.3.1 Important Files 363/43
  A.4 To Install the Peripherals 363/44
    A.4.1 Tape 363/45
    A.4.2 Printer (Kodak XL7700) 363/45
    A.4.3 Printer (Codonics NP600) 363/46
    A.4.4 Floppy 363/47
    A.4.5 Network to Other Hosts 363/47
    A.4.6 Serial Ports 363/47
  A.5 To Install Incremental Software Upgrades 363/47
  A.6 To Archive Data Files onto QIC-150 1/4" Tapes 363/48
  A.7 To Prepare an InstallSystem Tape 363/48
  A.8 PrepareSystemTape Script 363/49
  A.9 To Prepare an InstallApplication Tape 363/49
  A.10 PrepareApplicTape Script 363/49
  A.11 To Format the HardDisk 363/49
  A.12 To Build a New Kernel (Operating System) . . 363/50

A.13 BuildNewKernel Script 363/50

Appendix B 363/52
- B.1. Introduction: 363/52
- B.2. Screen spec: 363/52
- B.3. User_spec: 363/53
- B.4. Confocal_spec: 363/54
- B.5. Robot spec: 363/57
- B.6. Stage_spec 363/57
  - B.6.1. Procedure to determine stage_af_zero . 363/61
  - B.6.2. Procedure to determine stage_af_inten . 363/61
  - B.6.3 Procedure to determine stg_orthogonality_deg 363/62
  - B.6.4 Procedure to determine stage_angle_tangent 363/63
- B.7. Objective Specs 363/63
- B.8. Lon Specs 363/67
- B.9. System Specs 363/67

Appendix C: IRIS INDIGO BASIC COMMANDS 363/70
- C.1. Introduction 363/71
- C.2. Loggin in 363/71
- C.3. Help 363/71
- C.4. File manipulation 363/71
- C.5. File information 363/72
- C.6. Directory manipulation 363/72
- C.7. Find and search 363/73
- C.8. System information 363/73
- C.9. User information 363/73
- C.10. Command information 363/74
- C.11. Process control 363/74
- C.12. Time/Date 363/74
- C.13. File conversion 363/75

C.14. Piping commands 363/75
C.15. Sourcing and Sinking 363/75
C.16. Miscellaneous Commands 363/75
C.17. System shutdown procedure 363/76
C.18. Text editing using Jot 363/76
C.19. DOS Floppy Disk 363/77
C.20. Tapes 363/77
C.21. Adding users setup 363/78
C.22. Networking setup 363/78
C.23 IMPORT directory setup 363/78
    C.23-1 Network setup on the remote machine . 363/78
    C.23-2 Add the remote machine to LIS Network setup 363/79
    C.23-3, Make the IMPORT directory 363/79
    C.23-4 Make the IMPORT directory accessible by the remote machine 363/79
    C.23-5, Tell the system start using this new table: 363/79
C.24 REMOTE/REMOTE2 directory setup 363/79
    C.24-1 Network setup on remote machine . . . 363/79
    C.24-2 Add the remote machine to LIS Network setup 363/79
    C.24-3 Make the LIS REMOTE directory 363/80
    C.24-4, Enable network connection 363/80
C.25. File transfer 363/80
C.26. Logging on remotely 363/81
C.27. Home Directory 363/81

Appendix D: Support for KLA 20xx Defect File 363/82

Appendix E: Customer Preventative Maintenance Procedures 363/83
E.1. Scope: 363/83

E.2. Safety Precautions: 363/83
  E.2.1 Laser Safety: 363/83
  E.2.2 High Voltage: 363/86
  E.2.3 Moving Parts: 363/86
E.3. PM Schedule: 363/87
  E.3.1 System Startup 363/87
  E.3.2 System Shutdown 363/88
  E.3.3 Lockup Recovery 363/91
    E.3.3.1 Restarting the Ultrapointe Application 363/92
    E.3.3.2 Executing a Five Finger Reset 363/93
    E.3.3.3 Rebooting the Computer 363/93
    E.3.3.4 Cycling Power to the Instrument . . 363/94
  E.3.4 Wafer Removal 363/94
    E.3.4.1 Wafer Removal Using the Application Software 363/95
    E.3.4.2 Manual Wafer Removal 363/96
  E.3.5 System Wipe Down 363/97
    E.3.6.2 System Restore 363/98
  E.3.7 Chuck Cleaning 363/98
  E.3.8 Laser Lifetime and Interlock Inspection . . . 363/99
  E.3.9 Robot Alignment Check 363/101
    E.3.9.1 Robot to Cassette Alignment Check 363/102
    E.3.9.2 Robot to Pre-Aligner Alignment Check 363/102
    E.3.9.3 Robot to XYZ Stage Alignment Check 363/103
  E.3.10 Changing the Halogen Bulb 363/103
  E.3.11 Ultrapointe Factory PM 363/106
E.4. Preventative Maintenance Log Sheet 363/107
E.5. Problem Report Worksheet 363/108

Ultrapointe Model 1000
Laser Imaging System
System User's Manual
Rev. 1.3

1. Introduction

The Ultrapointe Model 1000 Laser Imaging System is designed to replace and out perform optical microscope review stations now utilized in the Semiconductor Fab environment to examine optical anomalies previously detected by Patterned Wafer Defect Scanning Instruments and to perform a variety of microscopic inspection functions. Significantly, the Ultrapointe system is the first defect review tool whose optics and functionality, have been designed explicitly for efficient performance of the dedicated production review and inspection tasks.

The Ultrapointe Model 1000 produces a three dimensional image, which may be rotated or tilted or shaded with correct perspective maintained with simple image rendering techniques without ever moving or disturbing the sample and which may be stored and recalled for later viewing. Also, the unit provides quantitative dimensional information (x, y, and z). The Ultrapointe Laser Imaging System has an ability the SEM cannot match, subsurface viewing of defects lying beneath dielectric layers. Combined with 3-D analysis software, the user is able to examine cross sections of the defect and surrounding material, and to assess the impact on the circuit layers.

The operator views the image on a CRT, with comfortable ergonomics, and the wafers are not be exposed to operator contamination or airflow. Unlike SEMs, the laser imaging system operates in air with class 1 cleanroom compatibility. While possessing metrological capabilities, the Laser Imaging System is not intended to compete with metrology systems now on the market, which also use confocal technologies, but are not designed for defect evaluation.

Cassettes of wafers (typically ranging from 3 inches (75mm) to 200mm diameter) are presented to the machine At the same time, a file of data from the wafer defect scanner, specifying the coordinates of the detected defects is transferred to the review station computer, either by diskette or other media, or by communication via a link or network such as Ethernet, RS232, etc.

A precision, high reliability robotic wafer handler removes wafers from the cassette and performs a prealignment step, sensing the notch and/or flat(s) on the wafer. The wafer is then loaded onto the optical unit's XYZ stage and is secured on the vacuum chuck. The operator accomplishes fine alignment ("de-skew") by lining up the visible light microscope field of view with either etched fiducial marks or other pre-specified structures on the wafer surface. Then, the unit can accurately translate any specified location on the wafer into the field of view with an accuracy of a few microns.

The operator examines each defect image as the tool presents it. The operator has three modes from which to select viewing. This examination can be either with the white-light conventional microscope optics, or in a real time laser scanning mode, or simultaneously with both laser and white light through the same optics. In white light, the operator can select from one of several objective lenses, varying effective magnification of the image. (The laser image scales simultaneously with the white-light image.) The white-light illumination may be polarized, and a cross polarized filter inserted in the observed light , causing optically active objects (such as quartz) to appear brighter. Optional bright-field-dark field objectives and illumination may be used to help locate particles and other defects on the wafer surface.

If the defect is not in view, or the operator wishes to examine a larger or different area, a joystick controller allows the operator to "cruise" the wafer.

Automatic operation features include autofocus, auto gain, and auto ranging of the vertical scanning. Engineering mode allows recipe setup to pre-specify operating parameters for use by operators working on a specific process level and product. Utilities are available in pop-up dialog boxes allowing manual control of the stage, polling of stage variables, plus robot manual control (e.g. allowing movement of a wafer from the flat finder or stage after a power failure). Diagnostics are called up via a pop-up window displaying all LON nodes and system variables, plus providing direct control of system functions.

By utilization of "recipes", the system may be pre-programmed for automatic operation. on a specific type of wafer, and at a specific process step for that wafer. The laser wavelength, and power, to be used may be selected. The number of slices of data and their spacing (in nanometers) may be specified. Finally, autofocus may be specified, and the offset in z (vertical) from the autofocus position to the ideal viewing position may also be preset.

A second major use of conventional microscope stations has been the more general inspection function, where pre-selected sites on a wafer are inspected for efficacy of a previous process step. The Ultrapointe Laser Imaging System is directly usable in this application, which faces exactly the same dilemma of defect imaging: Decreasing size of objects of interest, a lack of resolution, and the need for 3-D imaging.

Viewing through transparent layers allows determination of the vertical site of defects, or provide some specialized inspection. For example, stress voids in metal layers below dielectric layers may be viewed. The depth of metal plugs in glass insulators may also be seen.

2. Principles of operation

The system functional diagram (Figure 23) illustrates the general principles of operation. The Laser Imaging System uses the basic principles of confocal microscopy, where the illuminating light passes through a spatial filter (pinhole), and the image of this pinhole is then cast on the sample to be viewed. The light scattering from the sample returns through the optics to the pinhole, but only light from the focal plane of the imaging lens returns through the pinhole.

The light source is a multi-line laser (1) which produces polarized light at several discrete wavelengths which passes through a selectable filter (2) to isolate a single laser line or more than one line. A polarizing beam splitter (3) preferentially reflects light only of the proper polarization and directs it to the spatial filter (4). The spatial filter consists of optics which image the laser light on a pinhole whose diameter has been selected to re-image through the downstream optics and objective lens (5) producing a diffraction-limited spot. on the sample (6). The x-y beam scanner (7) consists of two mirrors which can oscillate their angle with respect to the beam in a manner to scan a raster pattern in space, when operated in conjunction with the scan lens (8). The raster scanned image of the pinhole is produced between the scan lens and beam splitter cube (9). In our first embodiment of this design, the slow scan is varied at 13 to 26 hertz, while the fast unit operates at 8 kHz. A raster scan of 256 or 512 lines is produced at approximately 26 or 13 frames per second, and is imaged at the back focal plane of the tube lens(11)

A quarter wave plate (10) is positioned to convert the linearly polarized laser light to circularly polarized light. The tube lens (11) works with the objective (5) to de magnify the image of the raster scanned pinhole image and project it on the sample. A portion of this light is scattered back into the objective and returns through the optic train. As it passes through the quarter wave plate (10), is converted to light linearly polarized at right angles to the light emitted by the laser. Thus when the light reaches the polarizing beam splitter (3), it passes through and reaches the photo detector (12).

As the raster is scanned, the surface data processor (13) keeps track of the returning intensity, and a two-dimensional intensity map of the sample is made. This image passes through the system computer (14) and is produced on the video monitor (15) as a digital image.

To obtain a three dimensional image, the height of the sample is varied by stepping the sample height small distances vertically using the fine z stage (16) between raster scans and keeping track of the vertical position for each horizontal point on the scan which produces the maximum intensity. This results in a 3-D map of the surface.

White light (conventional) microscope images are produced simultaneously with the live laser image by a video camera (19) which views the sample in white light emitted by the illuminator (20), an inserted in the optical path by the beam splitter (21). A filter which blocks the laser line in use but passes broad bands of light otherwise prevents the laser from saturating the image with reflected light.

A turret (23) allows objectives of different magnifying power to be used.

The optics unit must work in conjunction with a precision x-y stage (18) and "coarse" z stage (17). This module must accommodate from 3 inch (75 mm) to 200 mm diameter semiconductor wafers, working also with the robotic handler to load and unload the wafers with high reliability (< 1 wafer drop per million transfers), and class 1 compatible cleanliness.

The electronics package (Figure 24-1, 24-2, 24-3) contains all analog and digital electronics, plus power supplies, for complete operation of the tool. The unit operates on 220 v.(200-240 v. nominal), 50/60 Hz single phase electric power (or the European and Japanese equivalents).

The Surface Data Processor (SDP) (1) is a proprietary Ultrapointe design which interfaces with the photo detector and is synchronized with the scanner electronics, and fine z stage control, producing an x, y, & z map of intensity which can either be stored directly in the computer memory, or processed to extract a surface image on the fly. This interfaces through the SDP Interface (2) to the system computer (3).

The system computer (3) is a high speed RISC graphics workstation, capable of handling concurrent tasks of robot functions, stage motion, operator interface, and optics control, while also performing image processing functions. In addition, it functions with a windowing user interface and high resolution color graphics.

The X, Y, and coarse Z stage controllers (4) communicate with the system computer via an RS-232C interface (5), as do the robot and pre-aligner controllers (6).

The balance of the system electronic functions communicate through a Local Operating Network (LON) Interface (7) built on the same interface slot as the RS-232C interface. The LON itself (8) is a pair of wires that plug into each node serially around the system. Each node contains a local processor and firmware for LON communications, self diagnosis, and local operation of certain functions.

All user interface is via the system monitor, mouse/trackball, joystick/controller, and system keyboard. Through these controls and the windowing software, the operator can set up, program and operate the complete system, wafer selection and handling, defect editing and selection, automatic and/or manual wafer loading, defect classification, etc. In addition, he can utilize the tool as a microscope, selecting views, translating and focusing on details, changing magnification and zooming in on a sample. The operator console may optionally be remotely mounted. In addition, image processing and analysis functions may be performed from the console.

The overall system software is designed for operation on the review station in both operator and engineering mode, optionally using defect files supplied by various defect detection tools in a variety of formats. Both modes are password protected.

3. Safety

All user interaction with the instrument is via the front panel switches, the mass storage devices in the drive bay, or the keyboard and display console (see Figure 22). Servicing is to be performed by factory-authorized personnel only.

3.1. OPERATING SAFETY:

The Ultrapointe Model 1000 has been designed to minimize the hazards presented to the user during normal operation of the instrument. All servicing is to be performed by factory-authorized personnel only.

3.1.1. Laser Safety:

In accordance with the Regulations for the Administration and Enforcement of the Radiation Control for Health and Safety Act of 1968 (applicable to laser products), a Laser Product Report for this instrument has been forwarded to the Center for Devices and Radiological Health (CDRH). This instrument falls under Class I definitions because there is no user access to laser radiation.

CAUTION: Use of controls or adjustments, or performance of procedures other than those specified herein may result in hazardous radiation exposure.

The protective housing of the instrument should be kept intact during normal operation. Internal to the instrument is a Class IIIb argon-ion laser rated at 25 milliwatts with a maximum possible output of 500 milliwatts. Removal of any part of the upper protective housing can expose the user unnecessarily to hazardous laser radiation. Danger labels reproduced below indicate covers that provide access to hazardous laser light exposure.

**DANGER - Laser radiation when
open and interlock defeated.
AVOID DIRECT EXPOSURE TO BEAM**

**DANGER - Laser radiation when
open. AVOID DIRECT EXPOSURE
TO BEAM**

3.1.2. High Voltage:

Internal to the instrument are components using voltages up to 1000 VDC. Access to and servicing of all internal components is by factory-authorized personnel only.

3.1.3. Moving Parts:

As part of routine instrument operations, various parts of the machine make movements automatically and when commanded by the user. The user should take care to keep body parts and clothing away from the robot area. The user should also ensure that the current robot operation is complete before adding or removing a cassette from the cassette locator plate.

Much of the instrument is attached to an internal frame that is mounted to the external frame and protective covers via pneumatic vibration isolators. The "floating" internal frame can therefore move freely with respect to the covers and creates possible pinch points between floating components and the covers or external frame. Externally accessible floating components include the cassette platform, the robot, and the flat finder.

Inside the instrument, are many moving parts which could create possible hazards, including motors, shutters, and an X/Y/Z stage. Access to components inside the machine with system power on is restricted to factory-authorized service personnel.

Moving parts on the outside of the system include the robot handler, pre-aligner, and wafer shutter door, as well as the floating optics platform. The user should take care to keep body parts and clothing clear of all moving parts at all time.

4. Software Installation

Each Ultrapointe Model 1000 is pre-configured at the factory with all necessary system hardware and software. The system is ready to run after on-site installation by Ultrapointe Corp. No software installation is required.

However, for the purposes of system archiving and upgrades, each Ultrapointe Model 1000 is supplied with two 1/4 inch QIC-150 formatted tapes. One tape contains the generic computer operating system software, and the other contains the basic LIS software with some sample image files. See Appendix A for system backup, restore, and hardware upgrade procedures.

5. System Usage

5.1. Introduction

The Ultrapointe Model 1000 has a graphical user interface, and all functions can be accessed via "point-and-click" with the mouse. Within the LIS software, a screen is the collective look of the whole monitor screen, and it consists of a title bar, a top-menu bar and a collection of windows. The windows provide all primary functions, and the top menu bar provides access to the rest of the functions that are used infrequently. Windows are movable and sizable rectangles in various screens. A window encapsulates a major primary function and provides the necessary user interface. User can specify the arrangement of the windows in a given screen. Each item on the top menu bar pulls down a menu, and each item in the menu performs a specific task. The Model 1000 supports three levels of users: Normal user, administrator (engineering level), and service engineer. Normal users have no access to diagnostics and calibration functions.

5.1.1. Basic Controls

⇒ See Figure 25 for Basic Controls

5.1.1.1. Mouse

The red arrow on the screen will follow the movement of the mouse. By moving the mouse, one may position the red arrow over a widget such as a push-button, a menu bar item, an option menu, a radio button, a toggle button, or a slider bar.

Then, by clicking (pressing and releasing) the left button of the mouse, one activates the action associated with that particular widget.

5.1.1.2. Menu Bars

By clicking on a menu bar item, an associated pull down menu will appear. Then the user can activate one of the pull down menu items by clicking on it. Alternatively, one may press down on a menu bar item and not release the mouse button until the screen arrow is moved onto the desired pull down menu item.

5.1.1.3. Option Menus

An option menu is similar to a menu bar item, but it doesn't have to appear along the top of the screen, and it's the last selected item that will remain displayed. By clicking on an option menu, an associated pull down menu will appear. Then, the user can activate one of the pull down menu items by clicking on it. Alternatively, one could press down on an option menu and not release the mouse button until the screen arrow is moved onto the desired pull down menu item. The selected item will then remain visible as the other items in the pull down menu disappear.

5.1.1.4. Buttons

5.1.1.4.1. push-button

By clicking on a push-button, one would activate the action associated with the push-button.

5.1.1.4.2. Toggle

By clicking on a toggle button, one would turn the switch on, if it was originally off, or turn the switch off, if it was originally on.

5.1.1.4.3. Radio

Radio buttons are a group of toggle buttons where only one of the buttons can be 'on' at any one time. Therefore, by clicking on one button, the other buttons will all be turned off.

5.1.1.5. Slider Bars

One may move the slider bar within a scale by 2 methods:

1. By clicking on and dragging the slider bar to the desired value.

2. By clicking on the area in between the slider bar and arrow button. This will increment/decrement the slider bar by a value of 10. (Most scales have 10 as a default)

5.1.1.6 Entry Field

An **Entry Field** is a rectangular area where user can enter text and numbers. An Entry field accepts keyboard input only if it has the input focus, i.e.: the border of the entry field is a thick dark frame. Clicking within the entry field make the entry field to have input focus.

5.2. Data files and organization

The Model 1000 stores data(surface, volume, wafer map, recipe, and bit map image) in one of the following directories:

System	Accessible by all users
User	Normal user can access his own user directory
FLOPPY	DOS formatted 3.5" 1.44 megabyte floppy disk
OPTICAL	3.5" 128 megabyte read-write optical disk
IMPORT	Files to be imported to LIS in this directory. This directory is accessible from another machine. Linked via NFS(Ethernet, TCP/IP)
REMOTE	Physically located in another machine. Linked via NFS
REMOTE2	Physically located in another machine. Linked via NFS

A directory is created for each user, and this is the default directory where all data is stored for that user. A normal user can access only his own user directory, while an administrator can access all user directories. The ***System, FLOPPY, OPTICAL, IMPORT, REMOTE,*** and ***REMOTE2*** directories can be accessed by all users. All file names are keyed with a file extension. For example: all surface file names end with ***.sur*** and all volume files have names ended with ***.vol***. In the LIS, a directory listing shows only those file names with extension that matches the file type selected. The ***FLOPPY*** directory allows reading from and writing to MS-DOS formatted 3-1/2 inch floppies. Normally, the ***FLOPPY*** directory lists only those file names that match file type selected. If wafer map type is selected, however, all file names in the ***FLOPPY*** directory are listed.

The ***REMOTE***, ***REMOTE2*** and ***IMPORT*** directories are designed for interfacing to a data base host computer, for example, the Tencor Swift Station or the KLA 2550/2551. Some host computers require two directories for file transfer: The LIS receives a map file in the ***IMPORT*** directory, and the LIS sends the reviewed map file back to the host computer in the ***REMOTE*** directory. Other host computers need only one directory for both sending and receiving, and this directory is usually located in the host computer and accessed from LIS as the ***REMOTE*** directory. Generally, the ***IMPORT*** and ***REMOTE*** directories list only those file names that match the file type selected. If wafer map type is selected, however, all file names in the ***IMPORT***, ***REMOTE*** or ***REMOTE2*** directory are listed. As part of the installation, the ***REMOTE***, ***REMOTE2*** and ***IMPORT*** directories are configured to the correct network addresses. If the remote directories are not configured, or if the remote computer is not available, the ***REMOTE*** or ***REMOTE2*** choices are not displayed. The ***IMPORT*** directory is made accessible to another machine.

5.2.1. Volume and Surface

The basic image from laser scanning is a two-dimensional image at the focus plane. A volume data set is a series of 2D images taken at regular vertical intervals or "slices". From the volume data set, the system extracts the surface. The Model 1000 allows the user to view, manipulate, and store the surface as well as the volume data sets. Volume data set size is generally large (512 by 512 by 64 slices means 16 megabytes), and the corresponding surface data is generally smaller (512 by 512 by 2 bytes = 512 kilobytes.) Both volume and surface files are stored with their history records: date and time of image, wafer map used, operator, product ID, wafer slot, and defect number. If the image file is loaded from the top menu bar, the named wafer map is also loaded. Loading images from individual 2D or 3D windows will not update the wafer map.

5.2.2. Wafer map File

A wafer map file contains information relevant to the whole lot of wafers. Generally a wafer map file can handle a lot of two cassettes, and each cassette can have up to 25 wafers. The Ultrapointe 1000 supports many standard wafer map formats. Including Tencor^{tm}, KLA ^{tm}, and Inspex^{tm}, among others. Typical contents of wafer map files are:

5.2.2.1 Defect information

Each defect is specified in defect size, intensity, location, and classification code. Defect information is displayed in a list in the defect list window, and graphically in the wafer window.

5.2.2.2 Wafer information

Die size, flat orientation and wafer size are displayed in the wafer window. Deskew locations are displayed in the deskew window.

5.2.2.3 Identification

Product ID, lot ID and process ID are displayed in the status window.

5.2.3. Recipe

Recipes are used to customize and automate many of the instrument's functions. By loading a properly specified recipe, the user can setup the LIS with the exact screen layout and scan parameters, as well as defect sort criteria, without any user input. When a user logs in, the system will search the user's directory for a recipe with the user's name. If the recipe is found, it is loaded automatically. A recipe stores:

4.2.3.1 Screen layout

Different steps in the defect review process require different screen layouts for optimal results. Instead of manually arranging the screens, the user can design a number of screens in a recipe and select those screens as needed. Each recipe's screen is given a descriptive name, and the list of recipe's screen is accessible from an option menu at the lower right corner of the monitor. In addition to specifying the layout of screen's windows, the user can also specify the options within some of the windows, e.g.: 3D 2D and status.

4.2.3.2 Laser scan parameters

Most of the optics parameters can be specified in the recipe: Camera intensity, laser intensity, number of slices, scan dimensions, bright and dark field selection, camera filter, laser scan xy resolution, laser line, sample thickness, laser display color option, as well as stage autofocus intensities and stage-center-offsets. These parameters are loaded into the LIS when the recipe is loaded.

4.2.3.3 Defect Sort Criterion

Defects from a wafer map file can be filtered with a set of sort criterion so that only the desired defects are reviewed. The sort criterion can be specified in a recipe.

5.2.4. Bit map files:

Bit map images can be loaded and displayed in the library window. These are intended as reference images for defect classification. Bit map images are in Silicon Graphics RGB format.

5.2.5. Defect Code Table files:

The definitions of defect codes are stored in **Defect Code Table files**. User can edit, create, save and load these files.

5.3. Top Menu Bar

⇒ See Figure 25: Top menu bar

Clicking on an item on the top menu bar opens a menu of choices within that item. The Top menu bar provides the following groups of functions:

5.3.1. File:

The ***File*** item on the top menu bar provides access to load, save, and/or edit a variety of Ultrapointe 1000 files via a series of dialog boxes.

5.3.1.1. Image:

Popup the ***Image Open*** dialog box for loading previously stored volume and surface image files, and for saving current volume images. Note that surface images are saved from within a 3D window.

5.3.1.2. Recipe:

Popup the ***Recipe Edit*** dialog box for editing, loading and saving recipes.

5.3.1.2.1 Screen editing functions:

5.3.1.2.1.1 Add

5.3.1.2.1.2 Cut

5.3.1.2.1.3 Rename

5.3.1.2.1.4 Paste

5.3.1.2.1.5 Window swapping

5.3.1.2.2 laser Params

Popup optics parameters editing dialog box: Note that this dialog box is for editing the recipe's optics parameters, and does not control the parameters while it is being edited.

5.3.1.2.3 Sort Params

Popup the sort criterion editing dialog box.

5.3.1.2.4 File

Popup the file recipe file access dialog box. One can load and save recipe files.

5.3.1.3. Wafer Map:

Popup the Wafer Map ***FileSelection*** dialog box. In this dialog box, the user can:

5.3.1.3.1. Defect map file list

Display the list of available defect map files.

⇒ See Figure 26

The user can view the list of available maps from different directories. The System, Floppy and Optical Disk directories contain files that are available to all users. The ***User*** directory is the user's own directory. Each user generally can access his own directory, unless he is logged in as administrator, and in this case, he can select any user's directory.

5.3.1.3.2. Load the selected map.

Highlight the desired map from the list box and clicking on the ***load*** button loads the selected map. Any previously loaded map will be erased. If there is a wafer on the stage, the user will be prompted to unload wafer first.

5.3.1.3.3. Save currently loaded map.

Click on the ***selection*** text entry field and type in a file name to save the current map. The defect classification code is saved with the map.

5.3.1.3.4. Edit currently loaded map.

Some of the fields in a defect map can be edited: ***Wafer number, Lot ID***, and ***File Description.***

5.3.1.4.5. Clear Map:

Clear Map clears the current wafer map, and creates a new map. The user will be prompted for the desired Wafer Size (4, 5, 6, 8 in.), the Flat Orientation Angle (0, 90, 180, 270 deg.), and the Type of Map.

5.3.1.4. Defect Code:

Popup the ***Defect Code Edit*** dialog box for editing, loading, and saving the defect code set.

⇒ See Figure 27

5.3.1.5. Exit:

Popup the ***Confirm Before Quitting*** dialog box. Clicking the ***OK*** button allows the user to exit from the Model 1000 software. The **Quit** button cancels the command. Exit is available only to users logged in as Administrator or as Service Engineer.

5.3.2. Screen:

Screen Shows currently configured screen and allows the user to select from the list.

Clicking on a screen item brings up that screen.

5.3.3. Setup:

5.3.3.1. LaserScan Parms:

Popup the ***LaserScan Parms*** dialog box for controlling laser scanning. Most of the functions in this dialog box are the same as in a ***Laser*** window, except that Sample thickness, live laser color, stage autofocus parameters and Stage center offset can only be changed here.

5.3.3.1.1 Stage Center Offset

Set this parameter to make LIS1000 and scanner stage origin match. Setting this parameter correctly will ensure that the deskew points are within field-of-view of a 5X objective, thereby greatly ease the deskew process.

5.3.3.2. Deskew Options

Popup the ***Deskew Options*** dialog box . In this dialog box user can select the mode of deskew: Use Map Deskew Info, or Defects as deskew points.

5.3.3.2.1 Use map deskew info

In this mode, the deskew process requires user manually matching three predefined deskew points . These deskew points' stage coordinates are specified in the defect map file. This is the normal deskew mode for patterned wafers.

5.3.3.2.2 Defects as deskew pts

Bare wafers with defect maps can be reviewed by the LIS1000. On a bare wafer, there is no prominent pattern that can be used for deskew. Instead, the defects themselves can be used for deskew.

5.3.3.3 Edit Wafer Map:

Popup the ***Edit Wafer Map*** dialog box for changing current wafer map file. Changes made here are those that affect the whole file: wafer size, flat orientation, die size, deskew points and wafer IDs. Individual defect editing can be done from a ***Textstatus*** window.

5.3.3.3.1 Widgets:

5.3.3.3.1.1 FileName: shows the currently loaded wafer map file name.
5.3.3.3.1.2 Lot ID: allows editing of the LotId
5.3.3.3.1.3 Description: file description.

5.3.3.3.1.4 WaferSize option menu: allows choices of 100,125,150,200 mm

5.3.3.3.1.5 Flat Orientation option menu:

allows choices of Down, Left, Up, and Right.

5.3.3.3.1.6 Die Pitch:

X and Y die size in microns; not editable for Tencor formatted files.

5.3.3.3.1.7 Deskew1

First Deskew location, in microns; this field is not editable for KLA formatted files.

5.3.3.3.1.8 Deskew2

Second deskew location, in microns; this field is not editable for KLA formatted files.

5.3.3.3.1.9 Registration:

Displays the registration location. The registration location should NOT be changed unless it is known to be incorrect. Die patterns and defect locations are all relative to the registration point. Changing the location of the registration will shift the overall wafer map.

5.3.3.3.1.10 Wafer ID test entry field:

Allows user to enter wafer ID to specified slot. A valid wafer file must have at least one slot with a WaferID.

5.3.3.3.1.11 WaferID, Enter To List button:

Click on this button to accept the contents in the WaferID text entry field.

5.3.3.3.1.12 WaferID list box:

Displays WaferIDs for all slots. For a slot that has no map, "[No Map]" is displayed.

5.3.3.3.1.13 OK:

Accept changes as displayed and close the dialog box. All affected windows will update according to the new parameters.

5.3.3.3.1.14 Quit:

Ignore any changes made, and close the dialog box.

5.3.3.4. Defect List Sort:

Popup the ***Defect selection*** dialog box for editing defect sorting criteria. Sort criteria can be cascaded, and the Criteria list box displays the whole sort criteria.

5.3.3.4.1 Widget:

5.3.3.4.1.1 Criteria list box

Displays the whole list of sort criteria.

5.3.3.4.1.2 Criteria choice

Displays and allows selections of provided sort criteria.

5.3.3.4.1.3 Include and Exclude

Displays and allows selections of qualifier to the criteria. For the criterion ***ALL***, defects are always included and the choices of include and exclude do not apply.

5.3.3.4.1.4 Minimum

Inputs the minimum requirement for the criteria. For ***All, First,*** and ***Random*** criterion, the minimum requirement does not apply.

5.3.3.4.1.5 Maximum

Inputs the maximum requirement for the criteria.

5.3.3.4.1.6 Cut

Clicking on this button cuts away the selected criteria list item and copies it to the widget displays

5.3.3.4.1.7 Paste Up

Clicking on this button pastes criterion as displayed by the widgets to the criteria list box. New list box entries appear above the currently selected list box item.

5.3.3.4.1.8 Paste Down

Clicking on this button pastes criterion as displayed by the widgets to the criteria list box. New list box entries appear below the currently selected list box item.

5.3.3.4.2 Supported criterion :

5.3.3.4.2.1 All

Selects all defects from the wafer map.

5.3.3.4.2.2 First n,

Selects first n defects from the wafer map.

5.3.3.4.2.3 Random n,

Selects n randomly selected defects from the wafer map.

5.3.3.4.2.4 X size.

Selects defects with x size within the minimum and maximum from the wafer map.

5.3.3.4.2.5 Y size.

Selects defects with y size within the minimum and maximum from the wafer map.

5.3.3.4.2.6 Intensity.

Selects defects with intensity within the minimum and maximum from the wafer map.

5.3.3.4.2.7 Defect code.

Selects defects with defect code within the minimum and maximum from the wafer map.

5.3.3.4.2.8 X location.

Selects defects with x location coordinate within the minimum and maximum from the wafer map.

5.3.3.4.2.9 Y location.

Selects defects with y location coordinate within the minimum and maximum from the wafer map.

5.3.3.4.2.10 Every n

Select only one out of every n defects, i.e. for n=5, defect number 5,10,15,20.... are selected

5.3.3.4.3 Qualifier.

The criteria can be qualified with:

5.3.3.4.3.1 Include

The sorted list includes the defects that passed the sort criterion.

5.3.3.4.3.2 Exclude.

The sorted list excludes the defects that passed the sort criterion.

5.3.3.5 Rearranging Wafers

This dialog box is provided to rearrange the slot number of wafer maps in a wafer map file, in case that there is a mismatch between actual wafer placement and map file contents. Rearrangement is made only to the map file, no actual wafer movement is involved. Rearrangement is unnecessary if the wafer slot location matches those specified in the map file. In this dialog box, 25 buttons are provided corresponding to the 25 slots in a cassette. The slot with a wafer map is displayed in GREEN, slot with no map is colored WHITE. To move the wafer map from one slot to another: Click on the button for the first slot; hold down the mouse button while moving the mouse to the button of the second slot; release mouse button. The color of the button will change to reflect file changes. This Click-n-drag operation checks for the follow error conditions: Source slot has no map, destination slot already has a map, Click or release button on space other than the 25 slot buttons.

5.3.4. User:

5.3.4.1. Login:

⇒ See Figure 28

Popup the ***Login*** dialog box. This dialog box allows user to logout, so that others will have to log in to use the system, as well as log in as another user.

5.3.5. Config:

The ***Config*** menu is used for option configuration.

5.3.5.1. SECS:

SECS is planned to be used for the SECS II option, not currently installed.

5.3.5.2. Printer:

⇒ See Figure 29

Printer is used to setup the print options: Printer type, and number of copies.

5.3.5.3 Misc.

System Configuration dialog box allows Administrators or service engineers to edit the following system parameters:

5.3.5.3.1 Screen/laser saver timer, in minutes

This item specifies the number of minutes of mouse or key board inaction before the system goes to screen/laser saver mode. In the screen/laser saver mode, the Laser is switched to idle to prolong laser life, and the Laser window is turned to dark to save the monitor screen.

5.3.5.3.2 Deskew die Offset

For a defect file that contains no deskew information (i.e. KLA formatted files.) The LIS1000 generates the three deskew points at the lower-left-corner of three key dies. The left-most middle die is selected as the first deskew die, the right-most middle die is selected as the second deskew die, while the center, 00 die is used as the registration die. Sometimes, a wafer may have no pattern on the outer-most die locations, and the use of a non-existent die for deskew will cause problem. A ***Deskew Die Offset*** of 1 makes the dies next to the outer-most dies as the deskew die, an offset of 2 selects the dies 2 die space away from the left and right die.

The die offset adjustments applies even to those files that do have deskew information. The deskew points in a Tencor file, for example, sometimes point to an incomplete die, or a test die. In this case, a LIS1000 user can use the ***Deskew die offset*** to move the deskew points to another die.

5.3.5.3.3 Flat angle offset (deg)

If the wafer pattern is slightly skewed with respect to the wafer flat, or if there is an error in the robot teach angle, the wafer pattern on the monitor screen will appear to be tilted.

A stage movement along the wafer's horizontal street will cause the wafer pattern to move vertically. This XY tiling can be removed by adjusting the ***Flat angle offset.***

This offset parameter allows fine tuning of the angle in which a wafer is loaded onto the LIS1000 stage. The adjustment resolution is .125 degree(1/8 of a degree) and typical value is within +/- 0.5 degrees.

5.3.5.3.4 White light mode

Two white-light-only options are available: white light with a beam cube ***in*** or with it ***out***. The beam splitter is required in the laser scan mode to get simultaneous white light and laser viewing, as well as in autofocus. However, with the cube in, the white light intensity is cut by 50%(although image quality remains unchanged, and perceived image brightness is down by marginal amount.) For very dark samples, one therefore would want to switch out the cube in white-light-only mode. But, since the cube has to be ***in*** in the laser mode, the beam splitter cube will be switching in and out every time the laser and white-light-only mode is changed. The beam splitter switching is slow and can be a throughput bottleneck. To improve throughput, user has the option of keeping the beam splitter ***in*** in the white-light-only mode.

Cube Out:	brightest white light image, but switching between white-light and laser can be slow.
Cube in:	Slightly dimmer image for the same camera intensity, but otherwise no changes in image quality. Significantly better throughput and reliability.

5.3.5.3.5 Next die position

When viewing a defect, sometimes It is useful to compare the image with those at the same die location, at the next die. The LIS1000 has a button allowing user to go to the same location on the next die. User uses the ***Next Die position*** to specify where the next die is located: Bottom/Left/Top/Right.

5.3.6. Maintenance:

5.3.6.1. Manual stage control:

⇒ See Figure 30

Popup the ***Manual Stage Control*** dialog box. This dialog box is used to initialize the XYZ stage, as well as to perform stage diagnostics.

5.3.6.2. Manual robot control:

⇒ See Figure 31

Popup the ***Manual Robot Control*** dialog box. In this dialog box, users can initialize the robot, and command elementary robot functions. This is used, for example, to remove a wafer left on the stage because of a power failure.

5.3.6.3. File Transfer:

⇒ See Figure 32

Popup the ***File transfer*** dialog box to ***move, copy, rename*** and ***delete*** files.

5.3.6.3.1 File lists:

Files from two directories, selectable using the ***From*** and the ***To*** Option menu, are listed in this dialog box. File types listed are selectable from the ***FileType*** option menu.

5.3.6.3.2 File Type

The file type of a file is generally distinguishable from its file name extension, the file name's last three characters. The File transfer dialog box lists only one file type at one time, and user can choose surface, volume, wafer map, bit map, recipe, or defect code to list the specific file type. An ***All*** file type choice is provided for listing all files in any directory. When the Wafer Map file type is selected along with the ***FLOPPY, IMPORT,*** REMOTE or ***REMOTE2*** directory, then all file names are listed.

5.3.6.3.3 Target Name

Clicking on a file from the ***From*** directory selects that file for later delete, move, copy or rename. The name of the selected file is also copied to the ***TargetName*** entry field. The name in the entry field is used as the new file name in ***Rename***, and as the target name in copy and move. Users can edit the name in the entry field for the different name desired.

5.3.6.3.4 Delete

The selected file from the ***From*** directory is deleted.

5.3.6.3.5 Move

The selected file from the ***From*** directory is moved to the ***To*** directory with a file name from the ***TargetName*** entry field. The selected file in the ***From*** directory will be deleted.

5.3.6.3.6 Copy

The selected file from the ***From*** directory is copied to the ***To*** directory with a file name from the ***TargetName*** entry field.

5.3.6.3.7 Rename

The selected file from the ***From*** directory is renamed to be the name from the ***TargetName*** entry field.

5.3.6.3.8 File selection

User selects from the ***From*** file name list box by clicking on the desired file name. Multiple files can be selected: click on the first one, hold down the mouse button and drag the mouse to the last one. A warning message is displayed for deleting, coping, moving multiple files. Only one file can be selected for Rename.

5.3.6.4. UserAcct:

⇒ See Figure 33

Popup the ***User Accounting*** dialog box. Each model 1000 user is assigned a name, password (can be blank), and a privilege level (operator or administrator). Users can be added and deleted in this dialog box.

5.3.6.4.1 User Names list box

This list box displays the list of currently configured users.

5.3.6.4.2 Copy

Copies the user info of the high-lighted user in the user names list box. The **Name, Password** and **privilege level** will be updated.

5.3.6.4.3 Cut

Copies the user info of the high-lighted user in the user names list box. The **Name, Password** and **privilege level** will be updated. The high-lighted user will be deleted from the user list.

5.3.6.4.4 Paste

Create a new user and put into the user list, from the **Name, Password** and **Privilege** displayed.

5.3.6.4.5 Name, Password, Privilege level

Displays and Allows editing of user information. After editing a new user, the new user info needs to be pasted into the user list, and the user list must be saved.

5.3.6.4.8 Save

Save all the changes made, and the LIS software start using the new user list.

5.3.6.4.9 Quit

Ignore all changes to the user list, and quit the User Accounting dialog box.

5.3.6.5. LON Diagnostics:

⇒ See Figure 34

Popup the ***LON Diagnostics*** dialog box. This dialog box provides diagnostics and service control for most of the system hardware.

5.3.6.6. Calibration:

⇒ See Figure 35

Popup the ***Laser Scan Calibration*** dialog box. This dialog box is used to input the XY laser scan calibrations.

5.3.7. PrtScreen:

PrtScreen prints the whole screen on a digital printer.

It takes up to 10 seconds to capture the screen image. The user is asked to confirm printing when screen is captured, but before the image is send to the printer. If more than one digital printer is installed, the user is asked to specify the destination. The system will run slower than normal with printing in the background.

5.4. Window and Screen

There are eight window types in the Model 1000 graphical user interface: ***3D, 2D, Cassette, Wafer map, Status, Laser Scan, Image Library***, and ***uscope*** (microscope). These windows are sizable and movable and they may be configured by the user in a non-overlapping fashion. A specific **screen** is a particular arrangement of windows. The following screens have been pre-configured:

FourView
BigSurface
TwoSliceSurface
3View

Different screens can be installed by the user.

5.4.1. Common window attributes

⇒ See Figure 36

Each major functional module is encapsulated in a window. For some window types, one can have multiple windows of that type in a screen. For others, there must be one and only one of that window type in a screen. 2D, 3D, library and text status windows support multiple windows. There must, however, be one and only one of the following window type in a screen: Laser, microscope, wafer, and cassette. When switching screens, the contents of screen's windows carry over to the next if there is a corresponding window type in the new screen. It is more convenient therefore, to make all screens to have the same number and types of windows. For example, when one switches from a screen with three 2D windows to a screen with two 2D windows and back again, one of the three 2D window will lose its image. Generally, one would want to have three 2D windows to show the top ("XY"), side ("YZ"), and front ("XZ") views of the sample, two text status windows and two 3D windows. Although the functionality is different, all window types share the following attributes:

5.4.1.1. Window sizes.

All window types have three distinctive looks, depending on the size of the window. Size can be:

5.4.1.1.1. Large:

When the size is large enough to show the full functionality of the window. Typically a large window is wider than a third of the screen. All controls for a large window are directly accessible

5.4.1.1.2. Small:

When a window is large enough to show its major functions and a few controls but not large enough for full functionality. A small window allows access to its full range of functions indirectly through a control dialog box.

5.4.1.1.3. Icon:

When a window is less than 95 pixels wide. An iconized window displays only a button with the window name on it. To use the window, it must be swapped to a larger window location.

5.4.1.2. Swapable to other window sites on-screen.

All windows uses the click-drag method to swap between different windows.

Click and hold the mouse on the swap button in one window and drag it to another window. The two windows will interchange positions. The swap button is labeled "*L*".

5.4.1.3. Global and local data.

Operations originated from the top menu bar are global in nature, i.e.: they affect all windows. Operations originated from within a particular window affect that window only. For example, loading a surface file from the top menu puts the same surface on all 3D and 2D windows, as well as loading the wafer map file for that image. Loading a surface from within a 3D or 2D window updates the surface only in that window and will not update wafer map.

Volume data sets are generally large, and there is a limit of only one volume data set (file) loaded in the system memory at any one time. Loading a volume file from within a 2D window therefore also updates all 2D windows.

5.4.1.4. Inter-window data exchange.

There are some user interface features that require the cooperation of multiple windows; these features are implemented with the technique of inter-window data exchange. For example, the displaying of 2D XY cross sectional views using the top file load, allows the "***cut***" cursor in the XY display to control the slices viewed in the XZ and YZ displays in the other 2D windows.

5.4.2. Window types

5.4.2.1. Cassette Window

⇒ See Figure 37

The ***Cassette*** window provides the user interface for loading and unloading wafers in a cassette. It gives a graphical display of the current status of the wafer transport subsystem.

5.4.2.1.1. Wafer status:

There are three icons on the left of the cassette window, indicating the location status of a wafer among the robot arm, stage, and the pre aligner.

This window also displays the current cassette size, and the name of current wafer map file.

5.4.2.1.2. Cassette status and wafer selection

In the middle of the window, there are 25 rows of buttons and color bars corresponding to the 25 wafers in a cassette.

Buttons are for selecting the wafer, to be used for loading and unloading. In addition, prior to actually loading any wafer, one can also **preview** the different maps in the file, by selecting its corresponding slot. If the selected slot does not have a map, the user will be prompted if an empty map for that slot is to be created.

Color bars indicate the status of the cassette slot:

Present: A wafer is considered to be present in a slot on the cassette when that a wafer has been successfully loaded or unloaded in that slot.

Inspected: A wafer is considered to be inspected when it has been successfully loaded on the stage and returned (unloaded) back to the cassette.

File Present: Valid defect information for the slot is available from the currently loaded defect map file.

5.4.2.1.3. Command Buttons:

5.4.2.1.3.1 Lower25/Upper25

For a wafer map that has wafer information for 2 cassettes, this button Toggles between using the first/second cassette.

5.4.2.1.3.2. LoadWafer:

LoadWafer loads the wafer from the selected slot onto the stage. The load wafer sequence consists of:

Cassette presence check, chuck vacuum check, opening the wafer door, and moving the stage to the load/unload position.

The robot first takes the wafer from the specified cassette slot with its vacuum actuated end effector, and then transfers it to the flat finder (pre aligner).

The flat finder orients the wafer according to the specified flat position.

The robot retrieves the wafer from the flat finder, and puts it onto the stage chuck, where it is secured and sensed by the robot vacuum system.

5.4.2.1.3.3. Unload:

Unload removes the wafer from the stage and returns it to the selected slot. The unload sequence involves:

Robot picks up wafer from chuck, which has moved to the unload position.

Check for cassette status: Displays a warning message if the cassette has been removed.

Robot puts wafer into cassette slot.

5.4.2.1.3.4. Stop:

STOP stops current robot operation. A warning message is displayed and the user can ***reset*** the robot, or **resume** the original operation.

5.4.2.1.3.5. Initialize:

Reset initializes and reset robot to its reset position.

5.4.2.1.3.6. UsingMap/NotUsingMap:

A wafer map file can have multiple wafer maps: one for each slot. Two modes are provided for the different ways wafer map files are used.

In the ***UsingMap*** mode, loading a wafer from a slot with defect map information will also load that wafer map. If the map for that slot is not found, a new one is created. This new map will contain the same die information as the other maps in the file. In the ***NotUsingMap*** mode, wafer loading has no effect on current wafer map.

5.4.2.1.3.7. Flat orientation

This button tells the robot to orient wafer to one of the four directions: 0/90/180/270 degrees. "0" is with the primary flat/notch at the bottom, and the subsequent positions proceed clock-wise. The robot will use this new orientation to place the next wafer. Note: Most wafer maps will specify flat/notch orientation, and user normally needs not to change the wafer orientation from those specified.

5.4.2.2. Wafer Window

⇒ See Figure 38, Wafer Map Window

This window displays the name of the current wafer map file, wafer size, slot number and the XY coordinates of the current stage position, and selected target stage position.

There are two graphical display sections for the defects. The one on the left displays the overall view of the wafer. The one on the right displays a zoomed view of the section selected by the zoom box. The Zoomed view tracks the zoom box and update in real time.

5.4.2.2.1. Rubber band zoom box :

5.4.2.2.1.1. Zoom Box Size:

Click and hold on the lower right hand corner of the zoom box and then drag the mouse to change the size of the zoom box.

5.4.2.2.1.2. Zoom Box Positioning:

Click and hold on the upper left hand corner of the zoom box and then drag the mouse to move the zoom box.

5.4.2.2.2. Selecting a defect location

The wafer window is used to select an XY location to visit. In this window, there are two ways to select a defect location:

5.4.2.2.2.1. Graphical Selection.

Clicking on the desired position (defect sites are shown as colored dots) on the overview defect display or the zoomed defect display selects the location. Due to screen pixel size limitations, selecting a defect from the overview section will only be precise to several hundreds of microns. From a greatly zoomed display, however, one can generally reduce placement uncertainties to a few tens of microns.

Clicking on the ***GoTo*** button moves the stage to this selected position.

5.4.2.2.2.2. Type-in the desired location.

Type in the XY location on the XY text entry field, then click on the ***GoTo*** button to move the stage to this location

5.4.2.2.3. Operation without a wafer map

Without a wafer map, the wafermap window shows only the wafer outline without defect nor die grids. One can still use the wafer window, in this case, to move the stage and to read current stage positions.

5.4.2.2.4 Current XY stage position

Current XY stage coordinates are displayed in 2 ways: as numbers in the top left portion of the window, and as a cross in the graphic display. Current stage positions are updated at the end of all programmed stage moves. Also, they are updated when user clicks anywhere inside the overview defect display or the zoomed defect display .

5.4.2.3. Laser Window

⇒ See Figure 39: Laser Scan Window

This window provides the user interface for most of the white-light imaging control as well as the laser scanning functionality.

5.4.2.3.1 White light image

The live white light image is normally displayed on a separate monitor. The user has the option of selecting ***LaserScan*** or ***whiteLight***. In LaserScan mode, both the white light image and the laser image are displayed. The white light image is yellow filtered and thus appears yellowish. The WhiteLight mode gives the best white light image, but with only white light image displaying, the laser image is blocked,

5.4.2.3.2. Live laser imaging

The live laser scan image is displayed here. The user can use the joystick to move the stage in XYZ direction and watch for their effects on this window. When the laser window is large enough, a vertical scale showing the scan dimension is displayed on the right size of the Live Laser display.

5.4.2.3.3. Autofocus

There are two Z stages: the coarse and the fast.

The slower, Coarse Z has a travel of six millimeters, and is used to move the wafer surface into general focus from the load position of the stage (full-down).

The faster, fine Z has a travel of about 50 microns and is used for precision focus and to step the wafer through different Z positions during the volume and surface image acquisitions.

Two means of autofocus are provided: stage autofocus and fine autofocus.

5.4.2.3.3.1. Stage Autofocus

The ***Stage AF*** button initiates this autofocus: The fine Z stage is first moved to the middle of its travel, the coarse Z stage is moved three times at successively slower speed to find the best focus. A stage autofocus generally takes about 4 seconds. Coarse autofocus is generally used once, after the wafer is loaded, but may be required occasionally during inspection if the range of the fine focus is exceeded. Parameters for coarse auto focus can be pre-calibrated as part of the recipe, and should work for all samples, without regular user adjustments.

5.4.2.3.3.2. Fine Autofocus

The ***AutoFocus*** button initiates fine autofocus. The fine Z stage is first moved to its lowest position, then moves up until focus is within reach. A more detailed search of the best focus is then performed. A fine Z autofocus generally takes less than a second. The user may need to set the laser intensity for samples that are particularly dark or particularly bright. If fine autofocus fails, the Current Z slider goes to the top of its range (50 micron) or bottom of the range(0 micron).

5.4.2.3.4. Objective Selection

Click on the ***Objectives*** option menu to select desired objective. Several things happen when objective is changed: XYZ stage moves to effect parfocality and parcentricity, Laser intensity adjustments, camera attenuation for 5X objective and fine-Z-autofocus. All operations for objective changes are transparent to user, and is configurable by system administrator or service engineers.

5.4.2.3.5 Camera control parameters

5.4.2.3.5.1 Camera intensity

Adjusts between the nominal camera intensity between 0 to 255 for different samples.

5.4.2.3.5.2 Camera mode

Provides three options: normal(bright field), dark field and polarizer. Normal and dark field are applicable to both white-light-only and laser scan. Polarizer only works in white-light-only. If currently in laser scan mode, selecting polarizer camera mode switches system to white-light-only.

5.4.2.3.6. Laser scan parameters

These controls affect the operation of the laser and its scanner.

5.4.2.3.6.1. (Laser) Scan area

XY Laser Scan area can be selected by changing objective, XY scan resolution, and data size. The ***XY Res*** option menu displays the current laser scan XY resolution, and the horizontal and vertical range scales display the corresponding scan range.

A 20X objective gives e.g., 5 times the scan length and width as 100X objective. Click on the ***Objective*** button to select objective size.

Four XY Scan zoom settings, (corresponding to pixel sizes in laser display) are available: 100, 50, 25 and 12.5 nanometers (at 100X). The pixel size scales with objective magnification. Click on the ***XY Res*** button to select a pixel size/zoom factor.

Two data sizes are selectable: 512 and 256. Using the smaller data set requires less data storage and less processing and gives a faster response. Click on the ***Dimensions*** button to select.

5.4.2.3.6.2. Wave length selection

The Ultrapointe Model 1000 uses a multi-line argon laser for its laser source. User can select one of the three dominant wavelengths (458, 488, 515 nm), or all lines from the laser. (Additional argon lines are available as factory-installed options.) The ***LaserLine*** button allows wavelength selection.

In general, shorter wavelength gives better resolution, but there are several other reasons for selecting a particular wavelength. For example, sample spectral reflectivity, spectral opacity, and spectral absorption, all may play a role in selecting the wavelength.

5.4.2.3.6.3. Image intensity

Image intensity is affected by many factors: laser power, laser sensor gain, and sample reflectivity for the wavelength. The ***LaserIntensity*** slider control combines laser power and sensor gain in a normalized scale of 1 to 100. The user can use this control independently with the wavelength selection. The laser image sensor has a built-in protection mechanism that will shut itself down when intensity becomes too bright. The laser scan display becomes dark when the sensor shuts down, and the color of the ***LaserIntensity*** slider label changes to red. Clicking on the ***LaserIntensity*** slider bar at a lower value resets the sensor and the label color.

5.4.2.3.6.4. Focus change

Laser scanning has a very short depth of focus. By adjusting the ***Current Z*** slider control, the user can obtain a view of the sample at the desired depth. The fine Z stage control allows user to position the sample in steps of about 12 nm over a range of about 50 microns.

5.4.2.3.7. Image Positioning

5.4.2.3.7.1. Joystick

There is a toggle switch on the joystick, when this switch is in the up or "Z" position, moving the joystick up and down moves the coarse Z stage up (+Z) or down (-Z). In this position, the left-right motion of the joystick has no effect. The Z control with the joystick is intended for engineering use only. All routine focusing operation should be performed with the coarse and fine autofocus functions, and the fine z slider control. Crash protection firmware prevents accidental "crashes" of the objective against the wafer.

When the toggle switch is in the down or "XY" position, moving the joystick up and down moves the stage in the +Y and -Y directions and moving the joystick left and right moves the stage in the +X and -X directions. Joystick Z travel limits are set based on configurable sample thickness.

5.4.2.3.7.2. Point and go*

(Not yet implemented). Click on the screen commands the stage to move that point to the center of display.

5.4.2.3.8. Surface Imaging Mode

5.4.2.3.8.1. Principles

The underlying principle of scanning laser imaging is the short depth of focus of the confocal optics. By taking images at regular intervals of Z, the sample's XYZ intensity information is obtained. From this set of 3D volume data, the Model 1000 extracts the sample's 3D surface. In a 3D window, the sample surface can be viewed with different tilt and rotation angles, with zoom, different illumination effects as well as with different color schemes

A conventional microscope can only look at the top view of the sample, without Z information.

5.4.2.3.8.2. Slices

The number of slices selected is the number of data points in the Z direction. The Model 1000 has a scan speed of about

13 slices per second at 512 by 512 pixel image size and a maximum number of slices of 128. (At 256 by 256 pixels, the scan speed is 26 frames per second.)

5.4.2.3.8.3. Range Selection

The Z distance between slices is selectable to be between 1 to 30 basic steps, where a single step is approximately 12 nm. Use the ***Slices*** slider to select the number of slices desired. The number of slices and step size together determines the range of the Z motion in an image acquisition.

As a convenient feature, the user can use the ***S***(tart) and ***F***(inish) buttons to define the start and finish of a scan, and the system adjusts scan step size. The user utilizes the ***Current Z*** slider to move the laser focus above the region of interest. Clicking the ***S*** button sets the current z position as the starting point. Next, the user moves the slider down so that the laser focus is now below the region of interest. Clicking the ***F*** button sets the finishing point. Scan step size is then automatically adjusted to match this selected scan range, and the focus returns to the mid point between start and finish positions. Note that the ***S*** and ***F*** button are used for calculating the step size only. Actual data acquisition starts at half of the range above current Z position and steps at step-size for the number of steps specified.

5.4.2.3.8.4 SetZ

Instead of manually setting the Z scan parameters, the user can click on the ***SetZ*** button to automatically set the Z scan start position, range, and laser intensity. When user clicks on ***SetZ***, the system initiates a surface acquisition around the current Z over a 20 micron range. From the resultant surface, the system calculates and sets the best starting point, range and the optimal laser intensity. It also sets the fine Z stage to approximately mid point between the start and stop positions. The ***SetZ*** function takes about 3 seconds to complete for the 20 micron range. The actual range and time is configurable.

5.4.2.3.8.4.1Limitations of the SetZ function:

Although the ***SetZ*** function has been optimized for most typical use, there are cases where the Z scan parameters should be set manually:

The ***SetZ*** function searches for the sample's top and bottom within a range of about 20 microns. It can give erroneous results for defects taller than 15 microns.

The minimum range set by ***SetZ*** is about 2.5 microns (0.04 microns step size for 64 slices), and the resolution for this range may be too coarse for some flat defects.

5.4.2.3.8.5. Image Acquisition

Clicking the ***Surf Acq*** button starts the acquisition of a surface image at the half scan range above current Z position, using the pre-selected scan range. In this mode a volume scan is performed, and the surface data processor automatically produces the 3D surface.

5.4.2.3.9. Advanced features

5.4.2.3.9.1. Volume Data Acquisition

For a compositional complex sample, such as semi-transparent, multilayer structures, subsurface defects, etc., there may be a need to look at the basic volume data set directly, rather than the reduced surface. Clicking the ***VolAcq*** starts a volume acquisition scan without the surface extraction stage.

5.4.2.3.9.2. Multi surface analysis

Complicated samples with multiple layers can be viewed in the Model 1000. The following features have been implemented for extracting a particular layer surface:

4.4.2.3.9.2.1 Different view direction

Views of the volume data in 2D windows from different viewing directions.

4.4.2.3.9.2.2 New surface generation

From a 2D window displaying a XZ or YZ cut, generate a new surface with restricted start and finish slices.

5.4.2.4. Microscope window

⇒ See Figure 40

This window shows the white light video image in real time. It is not yet implemented.

5.4.2.4.1. Simultaneous Viewing

5.4.2.4.2. Objective Selection

5.4.2.4.3. Illumination Control

5.4.2.4.4. Image Filtering*

5.4.2.4.5. Bright field/Dark field Operation

5.4.2.4.6. Image Storage*

5.4.2.4.7. High-resolution Mode (no laser image)

5.4.2.5. 2D Image Window

⇒ See Figure 41

The 2D window can display surface images and volume images in any one of the 3 cutting planes: XY, XZ, YZ.

5.4.2.5.1. Multiple views within window

Within each 2D window, a sequence of 1, 4, 9, or 16 views can be displayed. Clicking the ***Sections*** button cycles through 1,4,9, and 16 views displays. The 1st Slice and Spacing slider bar determines the position of the first view and the interval between views.

Volume data size tends to be large in general, and there is a limit of one volume file in the system at a time. Because of this limit, loading a volume file, or acquiring a new volume data updates all 2D windows. No such limit exists for surface data.

5.4.2.5.2. Cursors

Clicking on anywhere within the view area brings up the cursor. There are two cursor types: sizing and cutting. Size and Cut buttons toggle between the two choices.

The sizing cursor is a "rubber band rectangle". Clicking and dragging on the upper left corner moves the cursor, clicking and dragging the lower right corner size the cursor. The vertical and horizontal dimensions of the cursor are displayed in the large screen window.

The cutting cursor is a "rubber band cross hair". Clicking and dragging on the horizontal or the vertical line moves the respective cursor line. The slices (cutting planes) addressed are displayed in the large screen window.

5.4.2.5.2.1. Multi-window operation, the cut cursor

For a complete 2D view of a 3D object, one probably wants to look from three different directions: top, side, and front. Having an indication of the positions of the cutting planes for the side and front view may also be useful. Rather than having a dedicated window having these complex functionality, the Model 1000 uses the concept of multi-window operation to achieve the same goal. For a complete 2D view, a Model 1000 user can use 3 2D windows, one for each cutting plane and selects the cutting cursor for the window displaying the XY view. The cursor information of this window is dynamically broadcast to other 2D windows. One window will therefore display the XY plane with the lines showing the cutting planes for the other two 2D windows.

5.4.2.5.3. Image appearance functions

5.4.2.5.3.1. Smoothing

Clicking the Smooth button smoothes the displayed data. If global surface data is displayed, displays in other windows may share this effect.

5.4.2.5.3.2. Color

Clicking the Color button toggles the display between false color and gray scale. (This function is different in the 3D window).

5.4.2.5.4. Viewing directions: Surface

5.4.2.5.4.1. Top view (XY)

In top view, the 2D window displays the surface in top view, in false color, or in gray scale. Since there is only one surface, displaying more than one view means display the same surface multiple times.

5.4.2.5.4.2. Vertical Slices (XZ, YZ)

A surface profile is displayed in XZ or YZ plane.

5.4.2.5.4.3. Multi-surface

More than one surface can be displayed in a 2D window, in XZ and YZ plane.

These surfaces may be generated from the same volume data using different surface extraction methods. Or, these surfaces may be similar but from different samples. Since each surface file holds only one surface, for multi-surface, all surfaces files must be loaded locally, one at a time, with the Add/Replace option set to Add.

5.4.2.5.5. Viewing directions: Volume File

5.4.2.5.5.1. XY, Top View

In top view, ***XY*** the 2D window displays the intensity volume data in projected top view, at the selected Z slice number.

5.4.2.5.5.2. XZ, front view

In front view, ***XZ***, the 2D window displays the intensity volume data as viewed from the front, at the selected Y slice number.

5.4.2.5.5.3. YZ, side view

In side view ***YZ***, the 2D window displays the intensity volume data as viewed from the left side, at the selected X slice number.

5.4.2.5.5.4. XZPeaks, YZPeaks

XZPeaks, ***YZPeaks*** are views from the same direction and at the same slice number as XZ and YZ. However, the extracted peaks, indicating possible surfaces, are displayed. The user may need to set the ***Threshold*** slider bar to cut out unwanted noise.

5.4.2.5.5.5. Z Profile

In the ***Z Profile*** view, the intensity profile along the Z axis at the selected XY location is plotted. This profile is useful as a diagnostics tool, as well as in multilayer viewing.

5.4.2.5.6. Image History

Surface and volume image are saved with their history records. Clicking the History button pops up the ***Image File History*** dialog box. The image file name, comments, date and time of acquisition, wafer map used, operator name, wafer slot, defect number, and Laser line used are displayed in the ***Image File History*** dialog box.

5.4.2.6. 3D Surface Image Window

⇒ See Figure 42

This window displays the surface in 3D. Various image manipulation tools are provided.

5.4.2.6.1. Image manipulation

5.4.2.6.1.1. Rotate

The ***Rot-*** button rotates the image counter clock wise, the + button rotates clockwise.

5.4.2.6.1.2. Tilt

The ***Tilt-*** button tilts the image toward top view, the + button towards the front view.

5.4.2.6.1.3. Zoom Function

Each click of the ***zoom+*** button brings the image closer toward viewer, so to make the image larger. The ***zoom-*** button reverse the effects of the previous zoom+ action. Click on ***Reset*** to undo zoom.

Detail control of Zoom is accomplished with the zoom cursor box, activated by clicking anywhere on the 3D window image. This is a special "rubber band box" whose size is controlled by depressing the mouse button, and sliding the mouse left or right. To position the re-sized zoom cursor, release the mouse button, slide the arrow cursor to the desired location and click once momentarily. The zoom cursor will move to the new location, centered on the arrow cursor position. Finally, click on ***Zoom+*** to implement the zoom.

5.4.2.6.1.4. Translation

The ← → and Up Down buttons move the image vertically and horizontally.

5.4.2.6.1.5. Z enhancement

Each click on the Z enhancement button enlarges the Z scale.

5.4.2.6.2. Image appearance

Surface data, height and intensity are displayed in the 3D windows in a photo-realistic algorithm. The image is rendered with illumination from a stimulated light source. The direction of the light source can be adjusted to appear increasingly from the left or right by clicking the ***Xdir-*** and + buttons, respectively.

5.4.2.6.2.1. Brightness and contrast

Clicking the ***Bri-*** and + button adjusts the overall image brightness.

Clicking the ***Contr-*** and + button adjusts the contrast.

Clicking the ***Color*** button repeatedly cycles through the four available color schemes:

- Height and intensity in false color,
- Height and intensity in gray scale,
- Height only, with height used as intensity, in false color
- Height only, no color or intensity information. Changing the lighting direction may be used here to

highlight small step changes, emulating Differential Interference Contrast

5.4.2.6.2.2. Smooth

Clicking the Smooth button smoothes the surface. Clicking Reset will not undo the smoothing effect.

5.4.2.6.3. Sectioning

Moving this ***Sectioning*** slider selects the ending point of the display, thus gives a sectioning effect at that point.

5.4.2.6.4. Reset

Clicking on the ***Reset*** button resets all the image manipulation steps to the default condition. Note however, the effect of smoothing cannot be reset.

5.4.2.6.5. Image History

Surface and volume image are saved with their history records. Clicking the History button pops up the ***Image File History*** dialog box. The image file name, comments, date and time of acquisition, wafer map used, operator name, wafer slot, defect number, and Laser line used are displayed in the ***Image File History*** dialog box.

5.4.2.6.6. Image Storage

A surface image can be saved by clicking on the ***Open*** button to popup the ***FileSelection*** dialog box. Click on the ***Selection*** text entry field, entering a file name, and then click on the

Save Surface button to save the displayed surface under that file name. The bit map of the currently displayed 3D image can also be saved by clicking the ***Save Bit map*** button.

5.4.2.7. Library Image Window

⇒ See Figure 43

The user can load and display previously saved bit map images in a ***Library*** window.

5.4.2.7.1. Library image selection

Click on the ***File*** button opens the ***File Selection*** dialog box. Highlight the desired bit map file name with the arrow cursor and click on the ***Load*** button to load the image.

5.4.2.7.2. Defect classification

In this mode, the general defect map information is displayed. The user can enter a description for the map, as well as a defect code for each defect visited. Clicking the ***List*** button pops up the defect list dialog box. When a defect map is saved, the entered description and the defect code is also saved.

5.4.2.7.2.1. Automatic use of defect library Function

The definition of the defect code is determined by the defect code file used. A defect code file can be created, edited, loaded and saved by clicking on the ***File*** menu on the top menu bar, and selecting the ***Defect Code*** item.

5.4.2.8. Text and status window

⇒ See Figure 44

This window displays text based status information. This window can be in one of four display modes: ***Defect status, deskew, defect selection,*** and ***wafer map edit***.

5.4.2.8.1. Defect status:

This mode is optimized for routine defect reviews. Defect code entry, defect selection and deskew can all be initiated here. The status window displays name of wafer map, product ID, Lot ID, process ID, number of defects displayed, number of total defects, operator name, defect site number, defect die number, defect XY coordinates.

5.4.2.8.1.1 Routine defect classification

Entering a defect code to the defect entry field tags the defect code to current defect and move the stage to next defect in the defect list. If another status window is visible showing the defect list, the defect code will be updated and the next defect will be high-lighted. Defect code entries can be made using only the numeric key pad, in ***NumLock*** mode. Pressing "/" key in the numeric key pad high-lights the defect code entry field.

5.4.2.8.1.2 EditDefect

The user can **append** a new defect or **remove** an existing defect. Bring up the Edit Defect Dialog Box by clicking on the

EditDefect push-button. To add a defect, type in its XY coordinates, its XY size, and its Intensity, and then click on the ***AppendDefect*** button. This defect will be appended to the Defect List. To remove an existing defect, select it in the defect list, and then click on the ***RemoveDefect*** button. This will change its Event Code to negative 1, which indicated this defect is not to be reviewed. Defects with their defect codes set to -1, will not be displayed if the Sorting routines are set to exclude defects with negative defect codes.

5.4.2.8.1.3 DeskewOff

5.4.2.8.1.4 CalcMapOffset

5.4.2.8.1.5 StartDeskew/ StartDefDeskew

See 5.4.2.8.2 for more detail

5.4.2.8.1.5 Prev Def

Click on this button to move the stage to the previous defect in the wafer defect list.

5.4.2.8.1.6 Next Def

Click on this button to move the stage to the next defect in the wafer map defect list.

5.4.2.8.2. Deskew

Deskew is the process of determining the amount of correction needed for pattern and wafer placement errors, so

RECTIFIED SHEET (RULE 91)

that the XY coordinates in the wafer map correspond precisely to the LIS1000 stage.

The deskew procedure consists of matching three reference points in the wafer map: the two deskew points and the one event origin point. Optionally, the user may use the offset feature to position defect to a particular part of the screen. After the deskew process, all stage coordinates are corrected to match those in the wafer map.

The LIS1000 system supports two deskew modes: patterned wafer deskew and bare wafer deskew. In a patterned wafer deskew, the deskew information from the wafer map is used to deskew. In a bare wafer deskew, the deskew information from the wafer map is not used, instead, user selected defects are used as deskew points.

5.4.2.8.2.1. DeskewOff

The deskew procedure involves many steps, any time within the procedure, one can click on the ***DeskewOff*** button to cancel the effect of deskew, so one can start the process over again.

5.4.2.8.2.2. Start Deskew(patterned wafer deskew)

Click on ***StartDeskew*** to start the normal deskew process. The XY Stage will be moved to the first deskew point, usually at the lower left hand corner of the first die from the left, on the middle row of the wafer. After autofocus, manually move the stage, with the joystick, until this lower left corner is at the center of the screen.

5.4.2.8.2.3. Accept Deskew1(patterned wafer deskew)

Click on the ***Accept Deskew1*** button to confirm the first deskew point and to move the stage to deskew point 2, usually at the lower left hand corner of the last die on the middle row. Manually move the stage with the joystick until this lower left corner is at the center of the screen. Generally, the amount of adjustment in the X (horizontal) direction will be small, perhaps up to few ten's of microns (within a quarter of the screen at 5X). An adjustment much larger than this indicates error, perhaps using the wrong die. The Y (vertical) direction will be larger, because wafer rotation is being corrected.

5.4.2.8.2.4. Accept Deskew2(patterned wafer deskew)

Click on the ***Accept Deskew2*** button to move stage to the reference event origin. This reference point is usually the middle of a street intersection, and is defined by the patterned wafer scanner used. Using the joystick, move this origin to the center of the stage.

5.4.2.8.2.5. Accept Deskew 3(patterned wafer deskew)

Click on the ***Accept Deskew3*** button completes the deskew process.

5.4.2.8.2.6 StartDefDeskew(bare wafer deskew)

Click on this button to start the process of bare wafer deskew. There is no stage movement, but the button's label changes to AcceptDsk1. The User is then expected to move the stage to a large (one micron or larger) defect from the defect list.

5.4.2.8.2.7 AcceptDsk1(bare wafer deskew)

After the user is sure that the defect displayed on screen is the one specified in the defect list, he can click on the ***AcceptDsk1*** button to accept that for the first deskew. After clicking the ***AcceptDsk1*** button, the stage will not move, but the label changes to ***AcceptDsk2***. The user is then expected to move the stage to another large (one micron or larger) defect from the defect list, preferably one that is a large distance to the right of the first.

5.4.2.8.2.8 AcceptDsk2(bare wafer deskew)

After the user is sure that the defect displayed on screen is the one specified in the defect list, he can click on the ***AcceptDsk2*** button to accept that for the second deskew. The deskew process is completed after the second deskew is accepted, and the button label changes to ***StartDefDskew***

5.4.2.8.2.9 SetDeskew1

For non-KLA formatted maps, the user can set the deskew points to current stage coordinates. This feature is useful in creating a defect map, setting the deskew points and then adding the defects.

Click on the SetDeskew1 button to define or reset the first deskew point to current stage position. A confirmation dialog box will pop up and the user needs to confirm to actually set the deskew1 position.

5.4.2.8.2.10 SetDeskew2

Click on the SetDeskew2 button to define or reset the second deskew point to current stage position. A confirmation dialog box will pop up and user needs to confirm to actually set the deskew2 position. Note that once the deskew1 and deskew2 are defined from actual stage position, the third deskew point (the registration point) is not needed for deskew. Although the deskew process still involves going through the three deskew points, the user should accept deskew3 without moving the stage.

5.4.2.8.3. Defect list

Display the ***Defect List***. The user can select a defect from this list and command the stage to view the selection, as well as view the next one on the list.

5.4.2.8.4. Wafer Map Edit.

Some of the fields in the currently loaded wafer map which can be edited are: ***Wafer number, Lot ID,*** and ***File Description.***

5.4.2.8.5 Search

A method of systematic spiraling search movement is implemented through this option. The user can specify the step size in each move: One die distance or 80% of the current field of view of the laser scan. The total area covered is through the spiral moves of a 5 by 5 move pattern. A set of 25 buttons are assigned for the corresponding search sections. Clicking on one of the 25 button moves stage relatively to that point. The button corresponding to the current search position is colored green, the rest white. Clicking on the ***StartSearch*** button advances to the next step in the search sequence. Clicking on the ***StartSearch*** button and holding down the button starts the search and continues the search, until the button is lifted. Clicking on the

ResetCenter button sets the current stage position as the center, the starting point of the search pattern, without any actual stage movements. All search movements are relative to current position, i.e. the user can independently use the joystick to move the stage.

5.4.3. Screen

A screen is a particular layout of windows: window type, size, and positions. A set of default screens are provided and can be selected from the ***Screen*** menu item on the top menu bar. When switching screens, the window contents are also transferred. No data reloading is required if the old and new screen have the same number and type of windows. All pre configured screens have the same number and type of windows:

Three 2D windows
Two 3D windows
One cassette window
One wafer map window
One laser window
One microscope window
One image library window
Two text and status windows

5.4.3.1. Switching to a different screen

There are two ways to switch to a different screen:

5.4.3.1.1. Select a screen from the *Screen* pull-down menu

5.4.3.1.2. Select a screen from the current recipe:

If a recipe has been selected, three buttons will appear at the bottom of the screen. These buttons select screens from the screen list in current recipe. ***Next*** selects next screen in the list. ***Last*** selects screen in front of current screen in the list. The middle button allows user to select from the list of available screens.

5.5. User privilege

The Ultrapointe Model 1000 software supports multiple user with different privileges and password. Each user has his own file directory where other normal user can not access. Three user privilege levels are supported: normal, administrator, and service engineer. Normal users are blocked from accessing functions such as calibration, and diagnostics. Administrators can access these functions, but are blocked from user account changes. Service engineer can access all functions. In the ***User Accounting*** dialog box, one can add, delete, modify users.

5.6. Operator Mode

The Ultrapointe Model 1000 should be started by a user with administrator or service privilege.

5.6.1. System Startup

The Model 1000 is configured so that on power up, the Model 1000 software is automatically started and ready for user

Login. Users are not expected to have to deal with the underlining operating system.

5.6.1.1. Laser imaging system Login:

⇒ See Figure 45: Copyright Screen

This copyright notice screen appears first and the user needs to click on the ***Continue*** button to go onto next step.

⇒ See Figure 46: Login dialog box

Click on one of the user names in the ***user name list box*** to select a user. If password is used, enter the password by clicking on the ***password entry field*** and then type in the password. Click on the ***login*** button to login as the selected user.

As shipped from the factory, only one user, "service" is installed with no password assigned.

5.6.1.2. Hardware initialization:

⇒ See Figure 47

Press ***OK*** to initialize Hardware dialog box.

After checking that the system is safe to start, click on the ***OK*** button. System hardware initialization takes several seconds. After hardware initialization, the manual stage control dialog box appears.

5.6.1.2.1. Stage initialization

Click on the **InitStage** button to initialize the XYZ stage. Stage initialization can take up to 20 seconds. After the stage is initialized, the user must approve homing of the X and Y axes. It can take up to 20 seconds to home all axes.

5.6.1.2.2. robot initialization

Swap in a ***Cassette*** window, if necessary. Click on the ***Initialize*** button in the cassette window to initialize the robot.

5.6.2. Wafer loading, cassette window

If a wafer map is available, it must be loaded before the wafer itself is loaded on the stage. To load a wafer map: click on the ***WaferMap*** item on the ***File*** menu. This pops up the ***wafer map file selection*** dialog box. Select the desired wafer slot in the ***Cassette*** window, and then click on the ***LoadWafer*** button. The robot will fetch the selected wafer from cassette, pre-align, and place it on the stage. The wafer map corresponding to the selected wafer slot is also loaded and displayed in the ***Wafer*** window.

5.6.3. Deskew, Text Status window

If a wafer map is used, the map's defect locations must first be corrected by the deskew process. The deskew process involves user visually matching 3 predefined wafer locations using the joystick. The first two are generally the lower left corners of the middle left and right die. The third deskew point is generally the lower left corner of the center die. The ***textstatus*** window provides the user-interface for the deskew process.

After the wafer is loaded, the system starts the deskew process automatically: The stage moves to the first deskew point, and autofocuses. The deskew button on the textstatus window displays ***AcceptDeskew1.*** After manually moving the stage to precisely the first deskew point, the user clicks on the ***AcceptDeskew1*** button. The system then moves to the second deskew point and the deskew button displays ***AcceptDeskew2.*** The user again moves the stage to match the second deskew point and then clicks on ***AcceptDeskew2.*** The deskew process is completed when the third deskew point is accepted. After the deskew process is completed, the system moves to the first defect on the wafer map.

5.6.4. Defect location, Wafer window

The user can easily select a defect location to review with the Model 1000. He can point-click and GoTo the defect in the wafer window's two wafer map views. Or, he can use the defect map list in the text status window to select a site in the defect list. In addition, entering a defect code to the current defect also moves the stage to the next defect. Once the site is selected and the stage has moved to that site, the system does an autofocus automatically. All imaging control functions, laser and white light, are provided in the ***Laser*** window. In the ***Laser*** window, there are two autofocus buttons: The fine autofocus and the stage autofocus. One starts with the ***stage autofocus,*** and then uses ***fine autofocus*** to bring sample into precise focus. The workings of the stage autofocus are pre configured, and no normal user adjustment is required. Fine autofocus has a range of about 50 microns and works over a wide range of sample intensity. However, if the sample is very dark or very bright, the user may need to adjust the laser intensity in the ***Laser*** window.

5.6.5. Image Acquisition, Laser window

Once the desired sample is brought into focus, it can be laser imaged. The Laser window can be in one of two modes: ***White Light Only*** to get the best white light camera image, and ***Laser and White Light*** for both laser scan and camera image.

5.6.5.1. White light only.

Select this mode when the best white light microscope image is desired. The laser shutter is closed, and only the white-light illumination falls on the sample. No filters are inserted except when selected by the user. Use:

5.6.5.1.1. *CamIntensity* slider to select camera image intensity,

5.6.5.1.2. *CameraMode* to select normal, dark field and *polarizer*.

5.6.5.2. White light and laser

In the ***White light and laser*** mode both the white light and laser image are available, and one can access all laser scan functions. The yellow filter is inserted in the white-light light path to prevent the laser light from saturating the video camera.

NOTE: When the image becomes much brighter than saturation, the laser image sensor may shut-down, causing the laser image to become dark green. Lowering the ***LaserIntensity*** slider setting will reset the sensor.

5.6.5.2.1. Setting scan range and scan start, manual method

Move the ***Current Z*** slider up and down and observe that the image intensity changes. The intensity displayed is adjustable with the ***LaserIntensity*** slider. Bright white color indicates saturation and dark green means no light sensed. On the brightest slice, the image intensity should be adjusted to be as bright as possible, without being saturated. Laser scan always starts down from the current fine Z position. Use the ***Slices*** and ***StepSize*** sliders to set the scan range and step size. Or, move the ***CurrentZ*** slider to the top of scan and click on the ***S*** button, then move the ***CurrentZ*** to the bottom of the scan and click the ***F*** button. The ***F*** button sets the ***StepSize*** slider for the selected slices to cover the whole range.

5.6.5.2.2 Setting scan parameters, automatic method

Clicking on the ***SetZ*** button starts the automatic scan parameter setting procedure. The system performs a search 20 microns around current Z, sets scan start, scan range, and optimal laser intensity. The ***SetZ*** command takes about 3 seconds.

5.6.5.2.3. Image acquisition

Clicking the ***StartSurface*** button starts the surface acquisition and clicking the ***VolAcq*** button starts the volume acquisition. In volume and surface acquisition, slices of 2D images are taken starting at current fine Z position and stepping down at *StepSize* interval for *Slices* number of slices. In volume acquisition, only 2D windows are updated with the new volume data, surface image is not generated. In surface acquisition, a surface is generated and displayed on all 2D and all 3D windows, but the volume itself is discarded.

5.6.6. Screen Printing

Clicking on the ***PrtScreen*** menu on the top menu bar prints the whole screen. After a screen is captured, taking about 10 seconds, a confirmation dialog box is opened automatically. The user can then click on the ***OK*** button to confirm the print or ***Quit*** to abort.

The number of copies and the type of printer can be adjusted from the ***Printer*** dialog box, opened by clicking on the ***Printer*** item from the ***Config*** Top menu bar.

5.6.7. Image storage

Both surface and volume data can be stored on the hard disk. Since the size of a volume data is generally large, one may want to limit the number of volume data files saved on disk at any time. Current volume data can be save by popping up the ***Image*** data file selection dialog box from the top menu. Surface data can be saved by popping up the image data file selection dialog box from a 3D window.

5.6.8 Defect Classification

A defect classification code is entered into the text status window's defect code entry field. After the code is entered, the stage moves to the next defect in the defect list. Windows displaying the defect list are updated with the entered defect code. An **Entry field** accepts keyboard input only if it has the input focus, i.e., the border of the entry field is a thick dark frame. Clicking within the entry field make the entry field to have input focus. Most of the defect code entry can be done using only the numeric key pad portion of the keyboard. Pressing the "/" key on the numeric key pad at any time forces input focus to the defect code entry field.

5.7. Engineering Mode

Engineering mode operations are those that deal with system configuration, recipe editing, calibration and maintenance. Most of these functions are restricted to administrators and service engineers.

5.7.1. User type and Operator access control

The Model 1000 defines three classes of users: regular user, administrator, and service engineer. Regular users has no access to other user's directory. The Administrator has access to all directories.

Regular users also are restricted from using most of the engineering functions.

4.7.1.1 User accounting

Clicking on the ***UserAcct*** item from the User menu on the top menu bar pops up the user accounting dialog box. From this dialog box, the Service Engineer can add or delete a user. A user's password and privilege status can also be changed here.

Changes will only take effect after being saved.

In case of error conditions, or for diagnostics, a user logged in as administrator can popup the manual control dialog boxes: the robot, and the stage manual control dialog box.

5.7.2. Recipe preparation

Many Model 1000's functions can be customized for the user and stored in a recipe file. A recipe with a file name the same as the user is automatically searched for in the user's directory when the user logs in. If found, this recipe is loaded and the system configure itself according to this recipe. Any Recipe can be edited by popping up the ***Recipe Edit*** dialog box.

5.7.2.1. Screen setup and sequencing

In the recipe dialog box, one can create a sequence of screens. The screen layouts are selected from a set of pre-installed screens. The user can swap the window in the screen. Clicking on the window brings up an ***Edit Window parameter*** dialog box where one can set the default conditions for that window. Current recipe screens are accessible via three buttons in the bottom screen bar

5.7.2.2. Laser parameter control

Laser scan parameters can also be customized in the recipe. Clicking on the ***LaserParams*** button brings up the ***Laser and Imaging control*** dialog box. One can set the desired laser scan parameters here so that the Model 1000 will come up this way when user logs in. A warning message: ***EditingOnly*** is displayed indicating no actual hardware is being controlled.

5.7.2.3 Sort criteria

All defects from the wafer map are filtered using the sort parameters. A number of sort criterion are provided, and they can be edited into the recipe. Clicking on the ***SortParms*** button brings up the ***SortParmetersEdit*** dialog box. In this dialog box, one can edit the sort parameters. See ***DefectList Sort*** menu item from the top Menu bar for more information on editing sort parameters.

5.7.3. Manual Robot control

The manual robot control dialog box is provided for single stepping the robot operation. The user can manually instruct the robot to get or put a wafer. Wafer source and destination are selected from the list box: stage, pre-aligner (flat-finder), and the cassette slots.

During robot movement, clicking the ***Stop*** button stops the robot. After the robot is stopped, a dialog box pops up so that user can continue the interrupted move, or reset the robot.

RECTIFIED SHEET (RULE 91)

5.7.4. Manual stage control

Manual stage control provides a list of elementary stage functions. All operations in this dialog box are in motor or encoder counts, except where noted. The user can use this dialog box to:

5.7.4.1. Enable/Disable joystick, all axes

5.7.4.2. Report Position/encoder count

5.7.4.3. Report sample thickness

Manually focus on the sample, then select the ***Z Dist Fr Config Limit*** function to display sample thickness. Since Z config Limit is the Z stage position at which the system is focused on the stage itself, Z Distance Fr Config Limit (when sample is focused) equals wafer thickness, in microns. This function assumes the maximum z travel has been set properly.

5.7.4.4. XYstage burn-in

The ***XYstage burn-in*** function commands the XY stage to move to random positions for a number of times. User specifies the number of times in the text entry.

5.7.4.5. Stop

This function stops all axes.

5.7.4.6. XYTo Load/unload

This function moves the XYZ stage to the loading/unloading position.

5.7.4.7. Move, absolute or relative

Move current axis to the specified count number, or to the specified number relative to current position.

5.7.4.8. Set speed

5.7.4.9. Set acceleration

5.7.4.10. Move to Home

Move current axis to its home position. Home for Z is at the z axis's full-down position.

Looking at the machine from the front, home for X is at the back, home for Y is at the right.

5.7.5. Hardware diagnostics

The ***LonDiagnostics*** dialog box is provided for subsystem diagnostics. In this dialog box, most of the common functions are directly accessible through slider or option menus. All functions are accessible through the Node and NetVariables list box.

5.7.6. System Calibration

The Laser scan dimensions can be calibrated by popping up the ***Calib*** dialog box. Clicking on the ***Calib*** item from the Maintenance menu on the top menu bar pops up the dialog box.

5.7.6.1. Laser scan XY dimension calibration

The model 1000 maintains a calibration table for the five objective's four scanner resolutions. This table can be calibrated by using the ***LaserScan Calibration*** dialog box.

5.7.7 System configuration

Some of the items in system configuration can be accessed from within the LIS1000 operating software. Currently, the following items can be edited in the Misc dialog box:

Laser saver timer
Deskew die offset
Pre-aligner (Flat finder) angle offset
White light mode
Next die position

Click on the ***Misc*** menu item from the ***Config*** top menu pops up the System Configuration dialog box.

5.7.7.1 Screen/Laser saver timer interval

Set the timer to the desired interval in minutes,

5.7.7.2 File DeskewDie Offset

Moves deskew points away default, The number entered determines the number of die deskew points are to be moved toward the wafer center.

5.3.5.3.3 Flat angle offset(deg.)

If the wafer pattern is slightly skewed with respect to the wafer flat, or if there is an error in the robot teach angle, the wafer pattern on the monitor screen will appear to be tilted. A stage movement along the wafer's horizontal street will cause the wafer pattern to move vertically as well. This XY tiling can be removed by adjusting the ***Flat angle offset.*** This offset parameter allows fine tuning of the angle in which a wafer is loaded onto the LIS1000 stage. The adjustment resolution is .125 degree(1/8 of a degree) and typical value is within +/- 0.5 degrees.

5.3.5.3.4 White light mode

Two white-light-only options are available: white light with a beam splitter cube ***in*** or with it ***out.*** The beam splitter is required in the laser scan mode to get simultaneous white light and laser viewing, as well as in autofocus. However, with the cube in, the white light intensity is cut by 50% (although image quality remains unchanged, and perceived image brightness is down by marginal amount.) For very dark samples, one therefore would want to switch out the cube in white-light-only mode. But, since the cube has to be ***in*** in the laser mode, the beam splitter cube will be switching in and out every time the laser and white-light-only mode is changed. The beam splitter switching is slow and can be a throughput bottleneck. To improve throughput, user has the option of keeping the beam splitter ***in*** in the white-light-only mode.

Cube Out: Brightest white light image, but switching between white-light and laser can be slow.

Cube in: Slightly dimmer image for the same camera intensity, but otherwise no changes in image quality. Significantly better throughput and reliability.

5.3.5.3.5 Next die position

When viewing a defect, sometimes It is useful to compare the image with those at the same die location, at the next die. The LIS1000 has a button allowing user to go to the same location on the next die. The user uses the ***Next Die position*** to specify where the next die is located: Bottom/Left/Top/Right.

5.7.8. System Startup

The Ultrapointe Model 1000 is configured so that on power up, the Model 1000 software is automatically started.

In the case where user may need to get out of the Model 1000 software for computer system usage, the following is provided as a guide to manually starting the model 1000 software.

5.7.8.1. Operating system and windowing system

The Ultrapointe Model 1000 software runs on the Silicon Graphics workstation over the IRIX operating system, SGI's version of UNIX. User interface is implemented using the X-Window windowing system, and Motif tool kit. The Model 1000 software is designed to provide all necessary functions without user having to interact with the underlining operating system or its user interface. However, IRIX also provides a graphical user interface, the WorkSpace, for system access, Please refer to the _IRIS WorkSpace User's Guide_ for more information on the ***WorkSpace*** and ***X-Windows*** windowing system.

5.7.8.2. IRIX Computer user autologin and program autostart:

IRIX is a multi-user operating system, and on power up, it will ask for user login. One can by pass this computer login step by assigning a user with Autologin option. A user "uv" has been installed, without password, but with autologin at the factory. This manual is written assuming you have logged in as "uv". Additional users and passwords can be setup using the ***System Manager*** tool from the ***Toolchest*** system menu on screen. After logging in, the *X Window* windowing system is started, and the ***toolchest, console icon,*** and the ***workspace window*** appear. The model 1000 is also configured at the factory to Auto-start the Model 1000 software. With autologin and auto start, the Model 1000 software runs after power up without user intervention. The file ***.sgisession*** in the user's home directory specifies the program to run when user logs in. Alternatively, especially after exiting from the Model 1000 software, double-clicking the mouse's left button on the ***uv1000*** icon in the WorkSpace window starts the Ultrapointe model 1000 system software.

6. Troubleshooting

6.1. Robot operation

The ***manual robot control*** dialog box is provided for single stepping the robot operation. The user can manually instruct the robot to get or put a wafer. ***Source*** and ***destination*** are selected from the list box: stage, flat finder, and the cassette slots.

During robot movement, clicking the ***Stop*** button stops the robot. After the robot is stopped, a dialog box pops up so that user can continue the interrupted move, or reset the robot. The Model 1000 communicates with the robot through a RS232. port. Robot can be disabled and assigned to different port by editing the ***/usr/uv/.uvconfig*** file. The positions of the robot arm are programmable and are stored in the robot controller. Please refer to the MECS robot user manual for detail.

6.2. Stage Operation

A list of elementary stage functions are provided in the manual stage control dialog box. All operations in this dialog box are in motor or encoder counts, except where noted. The Model 1000 communicates with the stage through the RS232 port. The stage can be disabled and configured by editing the ***/usr/uv/.uvconfig*** file.

6.3. Power fail Recovery

6.3.1. Wafer recovery

6.3.2. System component reset

6.4. LON diagnostics dialog box

All system functions can be accessed through this dialog box. Please use this dialog box with caution, as no safety check is made when accessing the system in this way.

6.4.1 Wafer door interlock.

Wafer door and laser shutter are wired together, so that when the wafer door is open, the laser is blocked.

6.4.2 Critical parameters

Some parameters work in groups, arbitrary resetting one without also changing the others in the group may cause damage to the system. The page scanner controller's Vertical Sync, TraceTiming, Retrace timing, StartAmp offset, Incr and Decr are all working together as a group.

6.5. System Service

6.5.1. Software configuration

All configurable software parameters are stored in ***/usr/uv/.uvconfig.***

See appendix B for details.

6.6. System Messages and Response

RECTIFIED SHEET (RULE 91)

6.6.1. Caution/Information

6.6.2. Error Condition and Recovery

6.6.2.1 Floppy disk not responding

The Model 1000 assumes the floppy drive is enabled, or "***Mounted***". If the floppy is unmounted, it will be inaccessible. Use the system manager in the Toolchest to check if floppy is mounted. See Appendix A.4.4 for floppy drive installation.

6.6.2.2 Printer not printing

6.6.2.2.1 SCSI connected printer, i.e.: Kodak XL7700:

The power-up sequence of the printer and the Model 1000 is important: the printer must be ready before the Model 1000. That is: make sure the printer is turned on first, wait till the front panel on the printer says it is ready, then turn on the Model 1000. The SCSI cable between the Model 1000 and the printer can be connected or disconnected only when both are turned off. If the printer has been off or the Model 1000 has been off, the order of power-up must be observed.

This printer is managed by the IRIX ***Print Manager,*** and printer status is available from the print manager. Printing is spooled by the IRIX in the hard disk. Do not queue up more than a few prints, as doing so may severely slow down the system, and may cause printer failure. Click on the system item on ToolChest and select ***Print Manager*** for status.

Trouble shooting: If after checking everything and the printer still is not printing, cycle power to the printer, wait till printer is ready, shutdown the Model 1000 and start again. The prints that are queued can be deleted in the ***Print Manage*** dialog box.

6.6.2.2.2 Network connected printer, i.e.: Codonics NP600

A network printer needs to have the correct address configured before it can accept print files. See Appendix A.4.3 for network printer installation. Once installed, the network print should need no special power-up procedure. Trouble shooting: If after checking everything and printer still is not printing, cycle power to the printer, wait for 3 minutes and print again. If the printer still is not printing, reset the print queue by re-starting the UV1000 and try again.

7. Applications Information

7.1. Wavelength Selection

The Model 1000 allows selection of different laser wavelengths in order to optimize viewing of the wafer surface. Selection depends on the spectral reflectivity, absorption, and transparency of the materials in the field of view, and the purpose of the observation.

The Model 1000 provides for selection of three primary argon ion wavelengths, 458 nm (violet), 488 nm (blue), and 515 nm (green). (Optional extra lines are also available). All give good resolution because they are at the shorter wavelength end of the visible spectrum, but the response of various materials and film thickness is quite different for each line. For example, polysilicon reflects the 515 nm line strongly, but not the 488 or 458.

The selection of wavelength also depends on whether the top surface is to be viewed, or whether a sub-surface feature is of interest. One value of wavelength almost invariably is most penetrating in a given situation, another is most reflected. Since materials and processes vary so much, experimentation is required in most cases to select the optimum. This information may then be entered into a recipe assigned to the product and process step of interest, and recalled by the operator during routine review and inspection operations.

7.2. Image interpretation

7.2.1. Surface reflectivity

The Model 1000 maps gray scale intensity into false color to enhance contrast and visibility of small differences in reflectivity.

7.2.2. Limits of performance

XY resolution and vertical resolution are a trade-off for very small structures. While the system can "see" individual 0.1 μm particles and resolve slightly larger structures horizontally, it cannot simultaneously resolve in the z (vertical) dimension until the x-y structure is on the order of 0.3 to 0.4 μm. When the x-y structure is of sufficient area $>0.5\ \mu m^2$, then the high resolution of the system vertically (approx. 25 nm).

The system in its simplest mode works on opaque, diffusely scattering surfaces.

Vertical surfaces are not well resolved.

Calibration latex spheres are a problem (as with all imaging optical instruments), because they act as tiny lenses, refracting the light and causing false images. Small, rounded transparent defects can have a similar effect..

Too small an object is a scatterer only, and shows up as an extinction object against a flat background.

Imaging down small diameter, high aspect ratio cylindrical holes of round , square, rectangular, or other cross sections can suffer when multiple reflections create false focuses.

Light scattering and interference effects can cause distorted images of sharp edges or high contrast change boundaries.

7.3. Complex surfaces

7.3.1. Transparent coatings

Many types of photoresist are transparent as are dielectric materials. These and other transparent media can cause confusion with the simple surface extraction process of the SDP. In the surface acquisition mode, the SDP selects the z coordinate at each xy coordinate which returns the brightest signal, and interprets it as a point on the surface. All surface points are connected to produce the display. If a (semi)transparent material covers a dark area, the brightest signal will come from the transparent surface. If it covers a highly reflective material, then the lower surface will be brightest and represent a surface point. If contrast varies on the lower surface, the SDP will produce an interpretation of the surface which may be an amalgam of regions of the two physical surfaces. In these cases, some of the advanced processing features may be needed to extract a single surface.

7.3.2. Multilayer structures

In the case of two or more layers to be viewed, the data should be taken with the ***VolAcq*** control, and viewed with the ***2D XZ*** or ***YZ*** views. The ***New Surface*** feature may be used to extract a single surface, or the ***XZPeaks*** or ***YZPeaks*** feature may be used to view the multiple surface profiles.

7.3.3. High contrast variations

Some surfaces, such as rough metal lines, will have a very large range of reflectivity in the field of view, which exceeds the dynamic range of the photo detector electronics. In order to see some of the lower contrast area, it may be necessary to increase ***Laser Intensity.*** This, in turn may cause the highest reflectivity portions of the image to saturate. If the saturation occurs over more than one step in Z, the SDP will be unable to select the brightest surface point at that XY location, and will treat the surface as lying at the first point where saturation occurred. This may distort the surface shape in the saturated region, which will appear white in color in the false color intensity display.

7.3.4. Vertical Surfaces

Virtually no reflected light is returned by vertical or near-vertical surfaces. While the surface algorithm shows a vertical surface, it is dark green to black in color, indicating no light was detected. In very dark areas, background noise may be detected by the surface algorithm, depicting a very rough, non-physical surface. Again, this is shown as dark green to black in color, indicating no light was detected.

8. System Maintenance

8.1. System software maintenance

8.1.1 Setting Date and Time

Exit Model 1000 software, and open a ***Winterm*** for a command prompt by clicking on the ***Tool*** item on the ***Toolchest*** menu and select ***shell.*** From the command prompt enter:

```
date mmddhhmm < enter>
```

example

```
date 06091415    < enter>   setting date/time to
      June 9,2:15
```

Close the Winterm by double clicking the icon on the top left corner of the window.

8.1.2. System shutdown

User must follow this system shutdown procedure before turning power off. Exit Model 1000 software, Click on the System item on the Toolchest and select System shutdown. A confirmation dialog box will popup for system shutdown confirmation: click Yes to shutdown or No to cancel request. The message " **Okay to power off the system now**" appears when the system is ready for power down.

8.2. Maintenance and Laser Safety

All maintenance procedures are to be performed with the laser power off. The system is equipped with interlocks to prevent accidental power-on status of the laser with the system covers removed. Never attempt to defeat the interlocks during normal use or maintenance.

8.3. Customer Preventative Maintenance Procedures

See Appendix E

8.4. Extraordinary maintenance

These procedures are to be followed in unusual circumstances.

8.4.1. Wafer removal

8.4.2. Other

9. Appendices

Appendix A

The ***InstallSystem*** Tape is for system software installation and the ***InstallApplication*** Tape is for application software installation.

One would install the SystemTape, then the ApplicationTape, and then the Peripherals.

A.1 To Install the Indigo System Software

1. If the Hard Disk needs to be Formatted, please see Section A.11
2. If the Hard Disk already has the operating system installed, Skip to Section A.2, otherwise continue.
3. The system normally has the TapeDrive already on the system's SCSI bus with a default SCSI ID of 7.
   Load the InstallSystem Tape into the TapeDrive.
4. Bring up the System Maintenance Menu by:
   Powering on the system.
   Pressing the Esc key, when prompted.
5. Load in the installation program by:
   Selecting Option 2 (***Install System Software***) at the System Maintenance Menu.
6. If the Hard Disk has just been formatted, the installation program will ask if it's OK to make New File Systems, Answer Yes, each time.
7. When the Installation Menu appears:
   Type 'install eoe2.man.bsdlpr'
8. Type 'install eoe2.sw.bsdlpr'
9. Type 'go'

10. After the 'Inst> prompt ' reappears, Type 'go' once again.
This is to install the maintenance software.
11. After the 'Inst> prompt ' reappears, Type 'quit'
12. Answer ***Yes*** to the Restart prompt.

A.2 To Install the Ultrapointe Application Software

A.2.1. Add a user named 'uv' by:

Logging in as '***root***'

Select ***System,*** in the ***ToolChestWindow*** at the TopLeftHandCorner of the screen.

Select the System Manager

Select the ***Users*** Icon.

Select ***Add.***

Type in 0000 for ***UserID.*** (This gives 'uv' superuser privileges).

Select ***Other*** for Shell

Type in '/bin/ksh' for the ***ShellName***

A.2.2. Bring up a ***ShellWindow*** by:

Selecting ***Tools,*** in the ***ToolChestWindow*** at the TopLeftHandCorner of the screen.

Select ***Shell***

A.2.3. Login as 'uv' by Typing 'login uv'

A.2.4. Load the InstallApplication Tape into the TapeDrive.

A.2.5. Type 'tar xv'. (This program will extract files from the Tape onto the HardDisk)

A.2.6. Login once again by Typing 'login uv'

A.2.7. Execute the command file (script) 'InstallApplic' by Typing 'InstallApplic'

A reboot will occur as the last step of the command file.

A.2.8. Turn on Autologin and turn on Workspace by:

Select ***System,*** in the ***ToolChestWindow*** at the TopLeftHandCorner of the screen.

Select the System Manager

Select the ***Users*** icon.

Select ***uv***

Select 'On' for Workspace

Select 'On' for Autologin

Select Accept

A.2.9. To start the Ultrapointe Application program via the autologin, just type 'reboot'

A.3 InstallApplic Script

```
# The following script assumes the printer's SCSI ID has
# been set to the default value of 6, and
# we are logged in as 'uv'
chmod a+rw /dev/ttyd1
chmod a+rw /dev/ttyd2
cp /usr/people/uv/Install/KodakXL7700.info  /usr/lib/print
cp /usr/people/uv/Install/k77print          /usr/lib/print
cp /usr/people/uv/Install/centface          /usr/spool/lp/model
cp /usr/people/uv/Install/mkcentpr          /usr/sbin
chown  lp   /usr/lib/print/KodakXL7700.info
chown  lp   /usr/lib/print/k77print
chown  lp   /usr/spool/lp/model/centface
chown root /usr/sbin/mkcentpr
chgrp   lp   /usr/lib/print/KodakXL7700.info
chgrp   lp   /usr/lib/print/k77print
chgrp   lp   /usr/spool/lp/model/centface
chgrp  sys  /usr/sbin/mkcentpr
chown lp   /dev/scsi/sc0d6l0
chgrp   lp   /dev/scsi/sc0d6l0
# cp /usr/people/uv/Install/hosts      /etc
cp /usr/people/uv/Install/printcap  /etc
cp /usr/people/uv/Install/bsdlpr     /etc/init.d
mkdir /usr/spool/np1
cp /usr/people/uv/Install/kernel  /usr/sysgen/master.d
mknod /dev/uldd    c 70 0
mknod /dev/ulsdp  c 71 0
cat  /usr/people/uv/Install/systemAddOn       >>
       /usr/sysgen/system
cp /usr/people/uv/Install/uldd       /usr/sysgen/master.d
cp /usr/people/uv/Install/ulsdp     /usr/sysgen/master.d
cp /usr/people/uv/Install/uldd.a     /usr/sysgen/boot
cp /usr/people/uv/Install/ulsdp.a  /usr/sysgen/boot
rm /etc/inittab.O
rm /etc/ioctl.syscon.O
rm /etc/passwd.N
rm /usr/sysgen/master.d/kernel.O
rm /usr/sysgen/system.O
cp /usr/people/uv/Install/.Xdefaults      /usr/people/uv
cp /usr/people/uv/Install/.sgisession   /usr/people/uv
cp /usr/people/uv/Install/.profile        /usr/people/uv
# cp /usr/people/uv/Install/uv                 /usr/people/uv
# cp /usr/people/uv/Install/.uvconfig     /usr/uv
autoconfig  -f
reboot
```

A.3.1 Important Files

/usr/people/uv/.Xdefaults	(X Window Config File)
/usr/people/uv/.sgisession	(Auto Login File)
/usr/people/uv/.cshrc	(User Config File)
/usr/people/uv/.login	(User Config File)
/usr/people/uv/.profile	(User Config File)
/usr/people/uv/uv	(Application Program)
/usr/people/uv/Install/InstallApplic	(InstApplicSoftwrScript)

/usr/people/uv/Install/InstallUpgrd	(InstUpgrdSoftwrScript)
/usr/people/uv/Install/PrepareSystemTape	(PrepSystemTapeScript)
/usr/people/uv/Install/PrepareApplicTape	(PrepApplicTape Script)
/usr/people/uv/Install/KodakXL7700.info	(Printer Install File)
/usr/people/uv/Install/k77print	(Printer Install File)
/usr/people/uv/Install/centface	(Printer Install File)
/usr/people/uv/Install/mkcentpr	(Printer Install File)
/usr/people/uv/Install/hosts	(Printer Install File)
/usr/people/uv/Install/printcap	(Printer Install File)
/usr/people/uv/Install/bsdlpr	(Printer Install File)
/usr/people/uv/Install/kernel	(Kernel Build File)
/usr/people/uv/Install/system	(Kernel Build File)
/usr/people/uv/Install/uldd	(Kernel Build File)
/usr/people/uv/Install/ulsdp	(Kernel Build File)
/usr/people/uv/Install/uldd.a	(Kernel Build File)
/usr/people/uv/Install/ulsdp.a	(Kernel Build File)
/usr/uv/.uvconfig	(Application Pgm Config File)
/usr/uv/data/*.bmp	(Bitmap Files)
/usr/uv/bruce/data/*.sur	(Surface Files)
/usr/uv/bruce/data/*.vol	(Volume Files)
/usr/uv/bruce/data/*.map	(Wafer Map Files)
/usr/uv/bruce/data/*.rcp	(Recipe Files)

A.4 To Install the Peripherals

All peripherals can be installed after the System Tape has been installed,except for the printer, which should be installed after both the System and Application Tapes have been installed.

A.4.1 Tape

A.4.1.1. The system normally has the TapeDrive already on the system's SCSI bus with the default SCSI ID of 7.

A.4.1.2. The TapeDrive is automatically installed when it's Detected on SCSI Bus.

A.4.2 Printer (Kodak XL7700)

1. Please note that the printer should be installed after the Application Tape has been installed.
2. Set the Printer's SCSI ID to 6 (Default)
3. Connect the printer to the system's external SCSI port. Power up the printer, 'Ready' should be displayed in the printer's LCD Window.
4. Reboot the system. Typing 'hinv' should output a line similar to

```
Printer: unit 6 on SCSI controller 0
```

5. Select System, in the ToolChestWindow at the TopLeftHandCorner of the screen.

6, Select the System Manager

7, Select the Printers Icon

8, Select Add

9, Type in a PrinterName (e.g. KodakColor)

10, If the printer is on the SCSI Bus, Select Parallel

11, Select the correct PrinterType, Select Accept

12, If the printer is on a remote Host, Select Network
    Type in the RemoteHostName and the RemotePrinterName,

13, Select Accept

14, Select the Printer that has been just added.

15, Select Yes for 'Accepting Requests'

16, Select Yes for 'Printing Requests'

A.4.3 Printer (Codonics NP600)

1. Please note that the printer should be installed after the Application Tape has been installed.
2. Install Files from Codonics Installation Diskettes onto the Printer by:

   Turning off the printer using the power switch at the BACK.

   Inserting Diskette #1 into the disk drive.

   Turning on the power. The printer will boot up and load in the software.

   Waiting until there are no more beeps (5 minutes).

   Ejecting the floppy and inserting the next diskette.

   Continuing this until all the diskettes have been installed.
3. Enter its IP address (or network address) by:

   Obtaining an IBM PC/AT keyboard.

   Turning off the printer using the power switch at the BACK.

   Plugging the keyboard into Port D at the back.

   Turning on the power using the power switch at the BACK.

   Turning on the power switch at the FRONT.

   Typing Control-C when the front display is flashing the word 'NETWORK'.

   Pressing ESC on the keyboard and waiting for 3 beeps. Redo this step if a typing mistake is made. Typing

   'i' <ENTER>

   Typing 127.0.2.200 <ENTER>

   Connecting the ethernet network cable to the COMM PORT.

   The word 'NETWORK' should be displayed (Not Flashing) on the front panel.

   Turning off the power using the power switch at the BACK

   Unplugging the keyboard.

   Turning on the the power using the power switch at the BACK.

A.4.4 Floppy

1. Select System, in the ToolChestWindow at the TopLeftHandCorner of the screen.
2, Select the System Manager
3, Select the Disks&Files Icon
4, Select Floppy
5, Type in '/FLOPPY' for the mount directory
6, Select Mounted
7, Select ShowDetailedInfo
8, Select Yes for 'Mount At Startup'
9, Select Accept

A.4.5 Network to Other Hosts

1. Select System, in the ToolChestWindow in TopLeftHandCorner of the screen
2, Select the SystemManager
3, Select the Networking Icon
4, Select Add,
5, Type in the New HostName
6, Type in the IP Address
7, Select Accept

A.4.6 Serial Ports

1. Select System, in the ToolChestWindow in TopLeftHandCorner of the screen
2, Select the SystemManager
3, Select Serial Port1, Set Port1 to Off
4, Select Serial Port2, Set Port2 to Off

A.5 To Install Incremental Software Upgrades

1. Load the InstallUpgrade Tape into the TapeDrive.

2. Execute the script 'InstallUpgrd' by:

Typing 'InstallUpgrd'

A.5.1 InstallUpgrd Script

```
# The following script assumes we are logged in as 'uv'
tar xv
```

A.6 To Archive Data Files onto QIC-150 1/4" Tapes

To archive or store Data Files on QIC-150 1/4" Tapes, please use the 'tar' command as described in Appendix C20.

A.7 To Prepare an InstallSystem Tape

1. Using a ReadyToRun Indigo XS24 (R3000), with a CDROM Drive and a TapeDrive attached:

Load a CDROM containing the operating system IRIX 4.0.5F. into the CDROM Drive,

Load a blank tape into the TapeDrive.

2. Mount the CDROM onto the directory /CDROM by:

Selecting System, in the ToolChestWindow at the TopLeftHandCorner of the screen.

Select the System Manager

Select the Disks&Files Icon

Select CDROM

Type in '/CDROM' for the mount directory

Select Mounted

3. Copy Files From the CDROM to a Tape via a Local Directory (/usr/TAPE).

Make sure have enough Disk Space under /usr by:

Typing: 'df'

Execute the script 'PrepareSystemTape' by:

Typing 'PrepareSystemTape'

A.8 PrepareSystemTape Script

```
# The following script assumes we are logged in as 'uv'
mkdir /usr/TAPE
cp /CDROM/RELEASE.info   /usr/TAPE
cp /CDROM/dist/eoe*      /usr/TAPE
cp /CDROM/dist/motif*    /usr/TAPE
cp /CDROM/dist/sa        /usr/TAPE
cp /CDROM/dist/mr        /usr/TAPE
distcp -vw  /usr/TAPE    /dev/nrtape
rm /usr/TAPE/*
rmdir /usr/TAPE
```

A.9 To Prepare an InstallApplication Tape

1. Load a blank tape into the TapeDrive.
2. Execute the script 'PrepareApplicTape' by:
   Typing 'PrepareApplicTape'

A.10 PrepareApplicTape Script

```
# The following script assumes we are logged in as 'uv'
tar cv /usr/people/uv /usr/uv
```

A.11 To Format the HardDisk

0. One would normally not have to do this.
1. Bring up System Maintenance Menu by:
   Powering on the system.
   Pressing the Esc key, when prompted.
2. Enter the Command Mode by:
   Selecting Option 5 (CommandMonitor) at the System Maintenance Menu.

3. Load in the DiskManagement program from the tape by:
   Typing "boot -f tpsc(0,7)(fx.IP12)"
   where the number '7' is the default SCSI ID of the TapeDrive.
4. When asked to be in ExpertMode, answer Yes.
5. When the DiskManagement(fx) Menu appears:
   Setup the Disk automatically by:
   Typing 'auto'. This will take approximately 30 minutes.
6. Exit from the fxMenu to the System Maintenance Menu by:
   Typing 'exit'
7. Then continue with Section A.1, Step 5.

A.12 To Build a New Kernel (Operating System)

0. Normally one would not have to build a new Kernel, unless there are new Device Drivers to be added in. Ultrapointe's 2 Device Drivers are already included in the current Kernel.
   They are for the GMI Board and the SDP Board.
   The GMI Board resides on the GIO Bus and has LON Network, and Serial Ports functionalities.
   The SDP Board has FrameGrabbing and SurfaceDataProcessing functionalities.
1. Execute the script 'BuildNewKernel' by:
   Typing 'BuildNewKernel'

A.13 BuildNewKernel Script

```
# The following script assumes we are logged in as 'uv'
# The following 6 commands have already been executed
# by the InstallAppl Script.
# They are included here for information purposes.
# cp /usr/people/uv/Install/kernel  /usr/sysgen/master.d
# cp /usr/people/uv/Install/system   /usr/sysgen
# cp /usr/people/uv/Install/uldd     /usr/sysgen/master.d
# cp /usr/people/uv/Install/ulsdp    /usr/sysgen/master.d
# cp /usr/people/uv/Install/uldd.a   /usr/sysgen/boot
# cp /usr/people/uv/Install/ulsdp.a  /usr/sysgen/boot
#
# Other Vendors commands should go here.
cd /
autoconfig -f
reboot
```

Appendix B

B.1. Introduction:

The file: */usr/uv/.uvconfig* contains all software configuration information. It is an ASCII file that can be edited by using the *jot* editor. Currently, the .uvconfig contains the following parameters:

B.1.1 screen spec
B.1.2 user spec
B.1.3 confocal spec
B.1.4 stage spec
B.1.5 robot spec
B.1.6 objectivesspec
B.1.7 turret step size
B.1.8 system specs

The general format of .uvconfig is:

B.1.2.1 Each line is at most 80 character long, characters after this limit are
ignored.
B.1.2.2 Lines that start with # are treated as comments.
B.1.2.3 Empty line is also treated as comment.
B.1.2.4 Every non-comment line is expected to be in the form of:
key_word = values

B.2. Screen spec:

line 0: screen_spec = nn
nn = number of lines that follows.

line 1: screen_name = name

name = descriptive name of the screen..

line 2... window_spec = WindowTypeKey xStart yStart width height

WindowTypeKey is one of the following:

- slices
- surface
- cassette
- rt_confocal <the laser window>
- referenceImg
- defect_map
- text_status
- microscope

xStart, yStart, width, and height are in screen pixels, in the range of x: 0 to 1262, y: 0 to 913. These are less than 1280x1024, because of the border and top/bottom displays.

Example:

```
screen_spec = 11
screen_name = 2Surfaces
window_spec = cassette    5   3  90  40
window_spec = surface     100 3  90  40
window_spec = microscope 5  46  90  40
window_spec = text_status 100  46  90  40
window_spec = slices        5  89  90  40
window_spec = rt_confocal  5 132  90  40
window_spec = defect_map 240  4  411  266
window_spec = surface      4  273  647  640
window_spec = slices     654    4  604    453
window_type = slices     654   460   604   453
```

B.3. User_spec:

line 0: user_spec = nn

nn = number of lines that follows

line 1+: name directory password privilege

name is the name of the user

directory is the name of the user's directory from /usr/uv

password can be alphabet or number, or a single '*' for none.

privilege is either 's' for administrator, or nothing for normal user.

Example:

```
user_spec = 2
bruce   bruce   abc     s
joe     joe     *
```

User specs can also be modified from inside of UV: click on the UserAcct item from the Maintenance menu.

B.4. Confocal_spec:

line 0: confocal_spec = nn

nn = number of lines that follows.

line 1: confocal_xy_res = aa bb cc dd

numbers are with 0.01 resolutions for the four scan resolutions aa,bb,cc,dd are nominal resolutions, after the machine is calibrated.

line 2: confocal_z_res = ee

actual fineZ step size, in nanometer

line 3: confocal_x_amp_obj0 = m n o p

line 4: confocal_x_amp_obj1 = m n o p

line 5: confocal_x_amp_obj2 = m n o p

line 6: confocal_x_amp_obj3 = m n o p

line 7: confocal_x_amp_obj4 = m n o p

line 8: confocal_y_amp_obj0 = m n o p

line 9: confocal_y_amp_obj1 = m n o p

line 10: confocal_y_amp_obj2 = m n o p

line 11: confocal_y_amp_obj3 = m n o p

line 12: confocal_y_amp_obj4 = m n o p

for all objectives, and page-scan XY amplitudes.

numbers are in integer, for the 4 scan resolutions, These are the actual page-scan x amplitude used to get the nominal XY resolution.

line 13: laser_pwr_at_line = xx yy zz nn

For each of the 4 laser line choices, the laser power lever is specified here.

xx = laser power setting for the 458 line, yy zz nn are for 488, 515, and all. range is 0 to 125 for 0 to 25mw.

line 14: laser_atten_at_458line = xx yy zz mm nn

line 15: laser_atten_at_488line = xx yy zz mm nn

line 16: laser_atten_at_515line = xx yy zz mm nn

line 17: laser_atten_at_Allline = xx yy zz mm nn

For each of the 4 laser line and 5 objective choices, the laser power filter selection is specified here.

xx = laser power attenuation filter position setting for the first objectibe position, yy zz nn are for second, third fourth and fifith. range is 1 to 6.

1 = 0 attenuation, 2 = minimum, 3 = 0, 4 = medium, 5 = 0, 6 = high attenuation.

line 18: laser_inten_dark_field_offset = xx

Compensate for the differences in laser intensity when switching between bright and dark field.

xx = (laser intensity at bright field) -

(laser intensity setting to get equivalent intensity at dark field) line19: pmtamp_min_offset_n_range = xx yy

xx is the minimum gain, can be -10 to 10. This is the gain sent to hardware if user selects 0 in LaserIntensity slider.

yy is the range of LaserIntensity slider. The LaserIntensity slider always display a range of 100, internally, 100 is linearly map into yy. yy can be between 80 to 120.

line 20: white_light_dark_field_power = nn

the number nn is the camera intensity used in dark field.

line 21: laser_white_switch_cube_1_0 = n

n= 0: the beam splitter cube is always in, no switching between laser/white light mode chnages.

n= 1: beam spliter is in for laser, out for white light. This setting degrade system throughput and reliablitiy.

Example:

```
confocal_spec = 21
confocal_xy_res = 100.00 50.00 25.00 12.50
confocal_z_res =  13.35
confocal_x_amp_obj0 = 1426  713   356  280
confocal_x_amp_obj1 = 1426  713   356  280
confocal_x_amp_obj2 = 1426  713   356  280
confocal_x_amp_obj3 = 1426  713   356  280
confocal_x_amp_obj4 = 1426  713   356  280
confocal_y_amp_obj0 = 1426  713   356  280
confocal_y_amp_obj1 = 1426  713   356  280
confocal_y_amp_obj2 = 1426  713   356  280
confocal_y_amp_obj3 = 1426  713   356  280
confocal_y_amp_obj4 = 1426  713   356  280
laser_pwr_at_line = 125 30 100 0
laser_atten_at_458line = 1 1 1 1 1
laser_atten_at_488line = 1 1 1 1 1
laser_atten_at_515line = 1 1 1 1 1
laser_atten_at_Allline = 1 1 1 1 1
laser_inten_dark_field_offset = -8
pmtamp_min_offset_n_range = 0 90
```

```
white_light_dark_field_power = 200
laser_white_sw_cube = 0
```

Confocal xy amps of all objectives can be calibrated from within UV:

Click on the calibration item from the maintenance menu. Confocal z resolution must be calculated manually and entered into .uvconfig with *jot.*

B.5. Robot spec:

line 0: robot_spec = 2

line 1: port_number = a

a = -1, 0,1,2,3 -1 = no robot, 0,1,2,3 = SGI port 1,2,3,4

line 2: 456inch_wafer_flat_size_deg = xx yy zz

The minimum wafer flat angle size, in degrees for the 4, 5 and 6 inch wafers. If one uses single flat wafer exclusively, xx,yy,zz can all be set to 0.5 degrees. If, however, two flat (major and minor) wafers are expected, then xx yy and zz should be set to angle that is half way between the major and minor flats.

Example:

```
robot_spec = 3
port_number = 2
456inch_wafer_flat_size_deg = 29.000 34.500 34.500
```

B.6. Stage_spec

line 0: stage_spec = 24

line 1: port_number = n

n= -1	no stage.
n= 0,1,2, 3	single rs232 port on SGI port 1,2,3, or 4.
n= 4	use 3 rs232 to communicate to the stage, SGI port 1/2/4 connected to X/Y/Z axis.

line 2: x_online = a

line 3: y_online = b

line 4: z_online = c

a,b,c= 0,1; 0 = offline, 1 = online

line 5: x_resolution = xx

line 6: y_resolution = yy

line 7: z_resolution = zz

line 8: x_encoder_res = xe

line 9: y_encoder_res = ye

line 10: z_encoder_res = ze

xx,yy,zz are micron per motor step, xe,ye,ze are micron per encoder step.

line11: max_z_um = aa bb cc dd ee

aa bb cc dd ee are the maximum stage autofocus z travel limits, in micron, for each objective.

line12: stage_af_zero = zz

xx is the dark level set during coarse autofocus

line13: stage_af_inten= aa bb cc dd

intensity values to autofocus for 458/488/515/All laser lines

line14: xy_200mm_level_zoffset = xx yy

xx yy are the Z psoition difference for a travel of 200mm, in micron.

xx = focus position(z) at Right of stage xy position - Z at Left

yy = focus position(z) at Top of stage xy pc ition - Z at bottom.

line15: xy_center_offset_um = xx yy

If the center of wafer is not exactly at stage coordinate 0,0, user can change this parameter to make it center.

line 16: joystick_gain = aa bb cc dd ee

XYJoystick gain settings for all objectives are specified here. Range = 1 - 5

line 17: joystick_z_gain = aa bb cc dd ee

Joystick gain of Z axis for all objectives. Range = 1 - 5

line 18: stage_angle_tangent = xx

Wafers are placed by the robot and the position of the wafer are programed into the robot. The robot position is coarse but is repeatible. Placement skew can be fine tuned with this parameter. The wafer rotation, in tangent is calculated: For the left and right points on a horizontal line(Pt1, pt2), xx = (Y2- Y1)/ (X2-X1)

line 19: stg_orthogonality_deg = xx

The angle between the stage's X and Y axes is machined into the stage and can not be changed. Adjustment of this angle, however, can be made in software to compensate for differences in tolerance. The degrees of XY orthogonality, expressed in degrees from 90 is specified here.

line 20 fine_af_offset = n

Diagnostics purposes, should always set to 0.

line 21: fine_af_algorithm = n

Diagnostics purposes, should always set to 0.

line22: stg_xy_init_final_spd_acc_um = xx yy zz xx is the initial xy speed, yy is the final speed, and zz is the acceleration of the xy axis for any programmed moves, all in microns.

line 23: stg_af_lo_mag_algorithm = x

x = 0: use first low-mag stage autofocus alogorithm, x=1: use second lo-mag stage autofocus. Second lo-mag stage autofocus requires fastZ firmware version 17(hex)

line 23: stg_af_lo_mag_algorithm = n

n can be either 0,1, or 2.

n = 0 old stage autofocus algorithm, to be used only for systems with fast Z board version 1.6 or older.

n = 1 new stage autofocus algorithm, this is a more reliable algorithm, but requires fast Z board version 1.7 or newer.

n= 2: no stage autofocus, instead, move to Z limit, less 150 microns.

Example:

```
stage_spec =  24
port_number = 3
x_online = 1
y_online = 1
z_online = 1
x_resolution = 0.2500
y_resolution = 0.2500
z_resolution = 0.1016
x_encoder_res = 0.5000
y_encoder_res = 0.5000
z_encoder_res = 0.1016
max_z_um   = 6950 6900 6950 6990 6500
stage_af_zero = 110
stage_af_parms = 58 40 40 40 40
xy_200mm_level_zoffset = 10 20
xy_center_offset_um = 100  -312
joystick_gain = 4 4 2 1 4
joystick_z_gain = 1 1 1 1 1
```

```
stage_angle_tangent = 0.0000000
stg_orthogonality_deg = 0.0000000
fine_af_offset  = 0
fine_af_algorithm = 0
stg_xy_init_final_spd_acc_um = 5000 150000 250000
stg_af_lo_mag_algorithm = 1
```

B.6.1. Procedure to determine stage_af_zero

B.6.1.1 Manually focus on a dark sample, with 100X objective at 458 line

B.6.1.2 Set laser intensity to just before PMT overload

B.6.1.3 Manually move sample away from focus.

B.6.1.4 Go to Ion diagnostic and select PMTAmp dialog box. Adjust AZOffsetCmd to a number that causes the laser image to be BLACK.

B.6.1.5 Back to laser window, manually focus sample again, adjust laser intensity to just before PMT overload. Redo step 6.1.3 and 6.1.4

B.6.1.6 Use the AZOffsetCmd setting for stage_af_zero, and the laser intensity setting for first number in the stage_af_inten.

B.6.1.7 Exit uv, enter the stage_af_zero, and 458 stage_af_inten into .uvconfig

B.6.2. Procedure to determine stage_af_inten

Note that the recipe also holds the stage_af_inten, and loading a recipe will overwrite current stage_af_inten the user login procedure includes a loading of the recipe with the user's name. The stage_af_zero is loaded only during program startup.

6.2.1 Start uv again, go to the LaserScanParm dialog box to select stage_af_inten for different laser line. Experiment with different settings to try to autofocus on dark and bright samples.

6.2.2 bring up the recipe's LaserScanParm dialog box from within the recipe dialog box, and enter the autofocus intensities.

6.2.3 Save the recipe.

B.6.3 Procedure to determine stg_orthogonality_deg

Set stg_orthogonality_deg to 0.000000 before start.

6.3.1 Obtain a test wafer with die patterns that are known to be orthogonal.

6.3.2 Move stage to the lower left corner of the middle left die, record actual position, Leftx = ______ , Lefty = ________ um

6.3.3 Move stage to the lower left corner of the middle right die, record actual position, Rightx = ______ , Righty = ________ um

6.3.4 Move stage to the lower left corner of the middle top die, record actual position, Topx = ______ , Topy = ________ um

6.3.5 Move stage to the lower left corner of the middle bottom die, record actual position, bottomx = ______ , bottomy = ________ um

6.3.6 tan(Horizonal) = $\frac{\text{RightY - LeftY}}{\text{RightX - LeftX}}$, Horizontal angle = _______ deg

TopX - BottomX

6.3.7 tan(Vertial) = -------------------------, Vertical angle

= ______ deg

TopY - BottomY

6.3.8 Orthogonality = vertical + horizontal, in degrees. Use this number to set stg_orthogonality_deg in /usr/uv/.uvconfig.

B.6.4 Procedure to determine stage_angle_tangent

This parameter is used to fine tune robotic wafer placement.

6.4.1 Load a test patterned wafer, move to the middle left side of the wafer.

6.4.2 Focus on the lower left corner of a die with 100X objective, write down the stage coordinates: LeftX= _____, LeftY = ______um.

6.4.3 Move to the lower left corner of a die in the right side of the wafer, staying on the same street. Stage coordinates at 100X, RightX = _____, RightY = _____.

$$\text{stage\_angle\_tangent} = \frac{\text{RightY - LeftY}}{\text{RightX - LeftX}}$$

B.7. Objective Specs

line 0: objective_spec = aa bb cc dd ee

aa,bb,cc,dd,ee are text labels for the name of the objectives

line 1: objective_power = p1,p2,p3,p4,p5

nominal power of the objectives

line 2: objective_na = n1,n2,n3,n4,n5

nominal NA of the objectives

line 3: objective_offset_nm = o1,o2,o3,o4,o5

moved after each objective change. The first objective is used as reference, so that o1 should be zero(0), Unused objectives should also be zero.

line 4: objective_xoffset_um = ox1 ox2 ox3 ox4 ox5

line 5: objective_yoffset_um = oy1 oy2 oy3 oy4 oy5

Stage XY moved after each objective change. The first objective is used as reference, so that o1 should be zero(0), Unused objectives should also be zero.

line 6: obj_Laser_int_458_offset = a458 b458 c458 d458 e458

line 7: obj_Laser_int_488_offset = a488 b488 c488 d488 e488

line 8: obj_Laser_int_515_offset = a515 b515 c515 d515 e515

line 9: obj_Laser_int_ALL_offset = aall ball call dall eall

The amount of relative laser intensity adjustment for objective changes at 458/488/515/ALL laser line. a458 should be zero(0).

Procedure to setting **obj_Laser_int_xxx_offset**, asumming objectives are arranged as follows: 5X, none, 50X, 100X, none:

B7.9.1 Load a typical sample, and focus on a typical feature in laser-scan mode with 5X objective at 458 laser line.

B7.9.2 Adjust laser intensity slider to get optimum illumination. Intensity=_______

B7.9.3 Change to different objectives setting and adjust intensity slider to get the same level of intensity. Record intenisty settings:

4585 = ____, 45850= ____, 458100= _____

B7.9.4 Change to 488 line and repeat intensity adjustments for all objectives.

4885 =_____, 48850=_____, 488100=_____

B7.9.5 Change to 515 line and repeat intensity adjustments for all objectives.

5155 =_____, 51550=_____, 515100=_____

B7.9.6 Change to All line and repeat intensity adjustments for all objectives.

All5 =_____, All50=_____, All100=_____

B7.9.7 set:

obj_laser_int_458_offset = 0 0 (45850- 4585) (458100-4585) 0

obj_laser_int_488_offset =

(4885- 4585) 0 (4885- 4585) (488100- 4585) 0

obj_laser_int_515_offset =

(5155-4585) 0 (51550- 4585) (515100- 4585) 0

obj_laser_int_all_offset = (All5- 4585) 0 (All50- 4585) (All00-4585) 0

line 10: objective_camera_int_all_offset = aall ball call dall
line 11: objective_camera_int_pol_offset = apol bpol cpol dpol
line 12: objective_camera_int_yel_offset = ayel byel cyel dyel
line 13: objective_camera_int_dim_offset = adim bdim cdim ddim

The amount of relative camera intensity adjustment for objective changes. aall should be zero(0)

Procedure to setting **objective_camera_int_xxx_offset:**

B7.13.1 Load a typical sample, and focus on a typical feature in white-light-only mode with 5X objective.

B7.13.2 Adjust camera intensity slider to get optimum illumination. Intensity=_______

B7.13.3 Change to different objectives setting and adjust intensity slider to get the same level of intensity. Record intenisty settings:

All5 = ____, All50= ____, All100= _____

B7.13.4 Change to polarizer and repeat intensity adjustments for all objectives.

Pol5 =_____, Pol50=_____, Pol100=_____

B7.13.5 Change to Yellow and repeat intensity adjustments for all objectives.

Yel5 =_____, Yel50=_____, Yel100=_____

B7.13.6 Change to Dim and repeat intensity adjustments for all objectives.

Dim5 =_____, Dim50=_____, Dim100=_____

B7.13.7 set:

objective_camera_int_all_offset = 0 0 (All50 - All5) (All100-All5) 0

objective_camera_int_pol_offset =

(Pol5- All5) 0 (Pol5- All5) (Pol100- All5) 0

objective_camera_int_yel_offset =

(Yel5-All5) 0 (Yel50-All5) (Yel100- All5) 0

objective_camera_int_dim_offset =

(Dim5- All5) 0 (Dim50- All5) (Dim100- All5) 0

line 14: cam_mode_norm_use_dim = aa bb cc dd ee

where aa,bb,cc,dd,ee are either 0 or 1, for each objective. 0 means using all-pass camera filter in normal camera mode, 1 means using dim camera filter in normal camera mode.

Example: :

```
objective_spec = 5     00    50    100    00
objective_power = 20 20 50 100 20
objective_na = 0.30 0.55 0.85  0.95  0.95
objective_offset_nm = 0 0 48400 48400 0
objective_xoffset_um = 0   0   -200 234 0
objective_yoffset_um = 0   0    120   99   0
obj_Laser_int_458_offset =  0 0 5 20 0
obj_Laser_int_488_offset =  0 0 5 20 0
obj_Laser_int_515_offset =  0 0 5 20 0
obj_Laser_int_all_offset =  0 0 5 20 0
```

```
objective_camera_int_all_offset = 0 0 20 30 0
objective_camera_int_pol_offset = 70 0 90 100 0
objective_camera_int_yel_offset = 20 0 40 50 0
objective_camera_int_dim_offset = 50 0 70 80 0
cam_mode_norm_use_dim = 1 0 0 0 0
```

B.8. Lon Specs

miscellaneous configuration for the LON network.

line 0: lon_spec

line 1: turret_step = xx

xx = turret ztep delay, set to 50

line2: filter_wheel_delay_laser_line = 600

line3: filter_wheel_delay_laser_atten = 600

line4: filter_wheel_delay_cam_filter = 600

Not currently used, set to 600

line5: cam_filter_ap_pos_5_or_6 = 6

use filter wheel position 5 or 6 as the ALL-PASS position. Using position 5 improves the camera filter wheel response time, but require new camera filter wheel firmware changes.

Example:

```
lon_spec
turret_step = 50
filter_wheel_delay_laser_line = 600
filter_wheel_delay_laser_atten = 600
filter_wheel_delay_cam_filter = 600
cam_filter_ap_pos_5_or_6 = 6
```

B.9. System Specs

line 0: system spec

line 1: laser_saver_minutes = xx

xx = the minutes of no mouse/key click before putting laser to idle, white light intensity to 0 and PMT gain to minimum. Any mouse click/key will restore previous settings

line2: kla_deskew_die_offset = xx
The two deskew points, deskew1 and deskew2 are moved in toward the wafer center by XX number of die.

line3: cfg_next_die_position = n

n = 0: next die is the one below the current die.

n = 1: to the left of current die

n = 2: to the top

n = 3: to the right

line 3:enable_bin_image_tx = n
n = 0: diable the saving of images in binary form.
n = 1: Enable the saving of images in binary form, in tiff format. Images in this form are in 8 bit gary scale image in top view.

line 4: zrange_search_num_steps = nn
nn = the number of steps in the ***SetZ*** procedure. default is 44 steps. The step size is fixed at 30 micro steps(approximately 0.5 microns per step.) Scan speed is about 13 steps per second.

line 5: zrange_search_laser_gain = x
where x is in the range of 7.0 to 13.0.

This is the fine tune adjustment mapping between laser intensity setting and the resultant image intensity. If this value is too low or too high, user may need to perform the SetZ function to get a reliable optimal laser intensity setting.

To check for zrange_search_laser_gain setting:

Set laser intensity to some low value so that image appears light green(about 10 for typical sample) at focus. click on SetZ, record the laser intensity setting. Click on SetZ again for 2 more times. If laser intensity is increasing after the second and third SetZ, then zrange_search_laser_gain is too low, increment by steps of 0.5

Example:

```
system spec
laser_saver_minutes = 5
kla_deskew_die_offset = 1
cfg_next_die_position = 3
enable_bin_image_tx = 0
zrange_search_num_steps = 44
zrange_search_laser_gain = 9.0
```

Appendix C: IRIS INDIGO BASIC COMMANDS

Contents:

C1. Introduction
C2. Logging in
C3. Help
C4. File Manipulation
C5. File Information
C6. Directory manipulation
C7. Find and search
C8. System information
C9. User information
C10. Command information
C11. Process control
C12. Time/Date
C13. File conversion
C14. Piping commands
C15. Sourcing and Sinking
C16. Miscellaneous Commands
C17. System shutdown procedure
C18. Text editing using Jot
C19. DOS Floppy Disk
C20. Tapes
C21. Adding users setup
C22. Networking setup
C23. IMPORT directory setup
C24. REMOTE/REMOTE2 directory setup
C25. File transfer
C26. Logging on remotely
C27. Home Directory

C.1. Introduction

The basic operating system, IRIX 4.0.5F, is already preloaded onto the Indigo. IRIX 4.0.5F is Silicon Graphic's version of Unix.

A list of basic command follows.

For information on such topics as Floppy Disk Setup, please refer to the Ultrapointe Model 1000 User's Manual, Appendix A.

There are also very useful information in the Online Manual Pages.

C.2. Loggin in

login "login_name"	login to the Unix system
passwd	change one's passwd

C.3. Help

man "command"	invokes online manual pages for a particular command
man -k "keyword"	search manual pages for a particular keyword

C.4. File manipulation

cp "srcfile" "destfile"

copy the file named "srcfile" to a file named "destfile"

mv "srcfile" "destfile"

move or rename a file named "srcfile" to a file named "destfile"

rm "filename" remove or delete a file.

can be very destructive, no deleted file recovery in Unix.

chmod "flags" "filename"

change permissions of a file

r = read, w = write, x = executable

u = user, g = group, o = other, a = all

chmod a+rw "filename"

give read and write permissions to everyone

chmod o-x "filename"

take away execute permission from others

chown "NewOwner" "filename"

ln "srcfile" "destfile"

instead of copying files, one can link files, and save space and redundancy.

"destfile" is becomes a pointer to "srcfile", so whenever one accesses "destfile", one is actually accessing "srcfile".

C.5. File information

more "filename"	same as MORE in DOS world, stop at each page
cat "filename"	same as TYPE in DOS world
head "filename"	show the first few lines in a file
tail "filename"	show the last few lines in a file
diff "file1" "file2"	show differences between files, line by line
cmp "file1" "file2"	compare two files, byte by byte
wc "filename"	show number of characters, words, and lines in a file

C.6. Directory manipulation

ls	list the contents of current directory "ls -al" to show dotfiles, permissions, ownership, dates, filesizes.
pwd	print working directory, (show current directory)
cd "directory"	same as CD in DOS world, change to another directory
cd ..	change to parent directory

cd /	change to the root directory
cd ~	change to your home directory
mkdir "directory"	same as MD in DOS world, make a new directory
rmdir "directory"	same as RD in DOS world, remove directory

C.7. Find and search

grep -i "keyword" *.c	search for the "keyword" in all files with extension ".c" while being case insensitive.
find "directory_to_start_search_from" -name "filename" -print	search for "filename" starting at this "directory"

C.8. System information

hostname	display the name of the this computer, eg. "irisdemo"
df -k	show amount of disk space free, (in kilobytes)
du -k	show amount of disk space used, (in kilobytes)
env	show your environment settings, such as PATH, SHELL, TERM, HOME, prompts, etc.
hinv	show hardware configuration, such as CPU type, memory size, serial ports, ethernet, disk drives, graphics board options, etc.
versions	show versions of installed software
uptime	show how long the system has been up, and the number of users logged on

C.9. User information

whoami	show whose account this is
who	shows who is currently logged on
finger "user_name"	show real name of "user_name", etc.

C.10. Command information

which "command"	show exactly which executable will be invoked when "command" is typed.
whereis "command"	show where the location of this "command" is, this includes its executable, source code, and manual page.
w	show which commands are being executed, and by which users, etc

C.11. Process control

reboot	Shutdown system and then Boot up again. No musical tune is played. To power down, choose the "System Shutdown" option on the "System" popup menu at the top, left-hand corner of screen.
Ctrl-C	kill current process
ps	show the processes running currently, along with their PID, (process identification number) "ps -e" to show every process on system.
kill "process_id_number"	kill the specified process identified by its PID, (process_id_number)
"command" &	execute "command in the background
fg	bring background process back to the foreground.

C.12. Time/Date

date	show the current date and time
date 06091445	change date and time to June 9, 2:45PM
time "command"	show the amount of time it takes to execute "command"
cal	show current month's calendar

C.13. File conversion

file "filename" show what type file this is,
ascii text, C program code, binary data, executable, image, archive, Postscript, packed, compressed, etc

to_dos "unixfile" "dosfile"
convert Unix file to DOS file, these are ascii text files convert <CR><LF> to <LF>, append Ctrl-Z (EOF_Marker)

to_unix "dosfile" "unixfile"
convert DOS file to Unix file, these are ascii text files convert <LF> to <CR><LF>, strip off Ctrl-Z (EOF_Marker)

C.14. Piping commands

"command1" | "command2"
use the output of "command1" as input to "command2"

diff "file1" "file2" | more
show the differences between "file1" and "file2", page by page.

C.15. Sourcing and Sinking

"command" > "filename"
put the output from "command" into "filename"

"command" < "filename"
use contents of "filename" as input to "command"

C.16. Miscellaneous Commands

su	become the super-user
Ctrl-D	quit being the super-user, can also be used to destroy a Terminal(BlueScreen).
RightMouseButton	press the right mouse button when in a

	BlueScreen to bring up a popup menu. Options in this Popup Menu will let the user change the font, or create another Terminal(BlueScreen).
Windows	All windows have the same basic functionalities, which are Move, Size, Minimize, Maximize, Raise, Lower, Quit. These functionalities can be activated either with the WindowManager's Popup Menu in the TopLeftHand corner or graphically with the mouse.

On the Ultrapointe Model 1000, we can also use the following commands:

ll	list the contents of the current directory with detailed info
hh	show the previous few commands
Ctrl-P	retype the previous command automatically

C.17. System shutdown procedure

0. Do not just turn off the power switch on the SGI Indigo computer.
1. Select the "System" menu in the top left hand corner.
2. Select the "System Shutdown" option.
   Do not choose the "Log Out" option.
3. Wait for screen message saying "OK to power off the system" appears, then turn off machine.

C.18. Text editing using Jot

Jot is an X Window application, easy to use, can cut and paste, 1 level of undo, search for a string, pull in a file.

jot "filename"	start the jot text editor. Use the left mouse button for selecting regions of text. Use the right mouse button for bringing up the popup menu.

RECTIFIED SHEET (RULE 91)

C.19. DOS Floppy Disk

cp "filename" /FLOPPY	copy a file from current directory to the Floppy Drive
cp /FLOPPY/"filename" .	copy a file from the Floppy Drive to current directory
eject	eject the Floppy Diskette from the Drive

C.20. Tapes

Tar is most universal type of tape drive format.

One usually copies files from the specified directories, onto a tape (archiving), or extracts files from the tape, back into their original directories.

tar tv	list the contents of this tape
tar cv "Filename1" "Filename2"	will format the tape to the tar format, then write the specified files onto the tape.
tar cv "Directory1" "Directory2"	will format the tape to the tar format, then write all files from the specified directories and their subdirectories onto the tape.
tar cv .	will write all the files from the current directory and its subdirectories onto the tape.
tar xv "Filename"	extracts the file "Filename" from the tape back into its original directory.
tar xv "Directory1" "Directory2"	extract all the files from the specified directories and their subdirectories from the tape.
tar xv	will extract all the files from the tape, back into their original directories, and subdirectories.

C.21. Adding users setup

1. Select the "System" menu in the top left hand corner.
2. Select the "System Manager" option.
3. Select the "Users" icon.
4. Select the "Add" pushbutton.
5. Enter login name
6. Fill in the detailed information such as password, full name, desired Unix shell. For Korn shell, select other, and enter /bin/ksh
7. Select Accept.

C.22. Networking setup

1. Select the "System" menu in the top left hand corner.
2. Select the "System Manager" option.
3. Select the "Networking" icon.
4. Select Add.
5. Type in the HostName of the workstation to be added.
6. Type in its IP address.
7. Turn on TCP/IP, turn off NFS and NIS.
8. Select Accept.

C.23 IMPORT directory setup

The IMPORT directory is a directory located physically in LIS but can be accessed by another machine as if it is its own directory. It is intended for another machine to IMPORT files into the LIS. Configuration:

C.23-1 Network setup on the remote machine

Obtain from network administrator the remote machine's name and its IP address, the LIS' name and IP address. Setup the remote machine so that it can access LIS' IMPORT directory. The LIS' IMPORT directory is `/usr/uv/IMPORT`.

C.23-2 Add the remote machine to LIS Network setup

see C22 Network setup for adding the remote machine

C.23-3, Make the IMPORT directory

make the /usr/uv/IMPORT directory with read/write access by:

```
mkdir  /usr/uv/IMPORT
chmod +rw  /usr/uv/IMPORT
```

C.23-4 Make the IMPORT directory accessible by the remote machine

Add the IMPORT directory to the export table, so it is accessible to other machine. Edit the /etc/exportfs file:

```
jot  /etc/exportfs
```

add the line:

/usr/uv/IMPORT

save the file

quit

C.23-5, Tell the system start using this new table:

```
exportfs -a
```

C.24 REMOTE/REMOTE2 directory setup

The REMOTE and REMOTE2 are two directories that LIS users can access in all file access dialog box. These two directories are intended to be mapped into directories that are physically located in different machines via TCP/IP and NFS. These directories are hidden until they are configured, Configuration:

C.24-1 Network setup on remote machine

Obtain from system administrator the remote machine's name and its IP address, i.e: host1 and 192.0.2.10, its directory LIS is expected to access. As an example, it is in **host1**, directory **/usr/xxx/yyy/zzz**. Make sure that this directory is exported to LIS with both read write access.

C.24-2 Add the remote machine to LIS Network setup

see C22 Network setup for adding the remote machine

C.24-3 Make the LIS REMOTE directory

make the /REMOTE directory with read/write access by:

```
mkdir  /REMOTE
chmod +rw  /REMOTE
```

C.24-4, Enable network connection

Edit the file /etc/fstab:

```
jot /etc/fstab
```

add the line:

```
Host:/usr/xxx/yyy/zzz    /REMOTE    nfs
rw,soft,intr,bg,0 0
```

save the file

quit

mount and use the directory:

```
mount /REMOTE
```

C.25. File transfer

There are 2 methods for transferring files to a remote system: rcp and ftp.

1. rcp hostname1:/usr/people/user1/filename1
   hostname2:/usr/people/user2/filename2

   note: if one is copying to the remote system, one must have permission to write into the remote directory.

2. ftp "hostname"

help	type "help" for a list of available commands
dir	list directory contents, can also use "ls"
get "filename"	receive a file
put "filename"	send a file
mget "*.c"	receive all files with the extension .c
mput "*.h"	send all files with the extension .h
cd "directory"	change to another directory on remote system
lcd "directory"	change to another directory on local system
pwd	show current directory on remote system
bye	disconnect from remote system

C.26. Logging on remotely

1. rlogin "hostname"
2. after being prompted for a login name and a password, it would seem as though one is logged on locally.

C.27. Home Directory

There are some dotfiles which need to be configured to one's taste. The home directory is the directory one is placed in upon loggin in. It is usually /usr/people/your_login_name

Dotfile are configuration files which are executed when one is loggin in.

Dotfiles are similar to autoexec.bat and config.sys in the DOS world.

Shells are powerful command interpreters. They are similar to command.com in the DOS world, but much more powerful. Can modify one's prompt, set aliases for long commands, recall previous commands, edit the command line, etc.

.login	for C shell, configuration file when logging in
.cshrc	for C shell, configuration file when new C shell is started
.profile	for Korn shell, configuration file when logging in
.Xdefaults	configuration file for X Window applications
.rhosts	permission file for remote logging in or remote file transfer

For example, if one wishes to let others perform a remote login (rlogin) or perform a remote file transfer (rcp), append the following lines to the file .rhosts

```
hostname1   username1
hostname1   username2
hostname2   username3
```

Appendix D: Support for KLA 20xx Defect File

The Ultrapointe Model 1000 supports the KLA 20xx series defect scanners with its ability to read and write 20xx formatted wafer files. Due to the 20xx series unique features, the model 1000 implemented the following user interface to automate defect review using 20xx series wafer maps:

D1 Wafer map contains no wafer size information, wafer size is set to the cassette size currently used.

D2. Toggling between test die and reference die is provided through a button in the textstatus window. This button moves stage alternately between defect locations calculated with test die and reference die.

D3. Actual defect location, in test die or reference die, is updated when user enters a defect code for the defect. When a defect code is entered in the textstatus window, the defect list window, and the wafer map window will display the changes.

Appendix E: Customer Preventative Maintenance Procedures

E.1. Scope:

These Preventative Maintenance (PM) procedures describe the maintenance operations on the Ultrapointe Laser Imaging System (LIS) that can be routinely performed by on-site customer personnel (referred to as customer service personnel). The operations are low-risk in the sense that they do not involve delicate adjustments or potential contamination of critical optical components. Furthermore, there is no requirement to bypass or defeat any of the safety interlocks of the machine. At all times, however, the service person is required to observe common safety practices and to avoid altering any adjustments inside the machine that do not pertain to the procedure at hand. These operations are intended to be supplemented by Ultrapointe factory PM operations performed on a less frequent basis.

E.2. Safety Precautions:

The Ultrapointe LIS Model 1000 has been designed to minimize the hazards presented to the user during normal operation of the instrument. The customer service personnel will not be exposed to hazards if the following procedures are followed carefully, but customer service personnel should exercise due caution and common safety practices when working on the machine. Refer to figure 1 for identification of the major components of the instrument.

E.2.1 Laser Safety:

In accordance with the Regulations for the Administration and Enforcement of the Radiation Control for Health and Safety

Act of 1968 (applicable to laser products), a Laser Product Report for this instrument has been forwarded to the Center for Devices and Radiological Health (CDRH). This instrument falls under Class I definitions because there is no user access to laser radiation.

CAUTION: Use of controls or adjustments, or performance of procedures other than those specified herein may result in hazardous radiation exposure.

The protective housing of the instrument should be kept intact during normal operation. Internal to the instrument is a Class IIIb argon-ion laser rated at 25 milliwatts with a maximum possible output of 500 milliwatts. Removal of any part of the upper protective housing can expose the user unnecessarily to hazardous laser radiation. Danger labels reproduced below indicate covers that provide access to hazardous laser light exposure.

DANGER - Laser radiation when
open and interlock defeated.
AVOID DIRECT EXPOSURE TO BEAM

DANGER - Laser radiation when
open. AVOID DIRECT EXPOSURE
TO BEAM

Customer service personnel will be required to remove the top protective cover in order to perform some of the procedures described here. The cover is interlocked with the laser power supply so that laser radiation is not present after the cover is removed. **The top cover interlocks are defeatable but should not, under any circumstances, be defeated by customer service personnel.**

E.2.2 High Voltage:

Internal to the instrument are components using voltages up to 1000 VDC and 240VAC. Where the procedures in this document permit the customer service personnel to have access to components that may be at hazardous voltages, there will either be warnings or instructions to shutdown, turn off, and lock out the AC input to the machine. In this manner, the customer service personnel will not be exposed to hazardous voltages within the instrument.

E.2.3 Moving Parts:

As part of routine instrument operations, various parts of the machine make movements automatically and when commanded by the user. The user should take care to keep body parts and clothing away from the robot area. The user should also ensure that the current robot operation is complete before adding or removing a cassette from the cassette locator plate.

Much of the instrument is attached to an internal frame that is mounted to the external frame and protective covers via pneumatic vibration isolators. The "floating" internal frame can therefore move freely with respect to the covers and creates possible pinch points between floating components and the covers or external frame. Externally accessible floating components include the cassette platform, the robot, and the flat finder.

Inside the instrument, are many moving parts which could create possible hazards, including motors, shutters, and an X/Y/Z stage. Where the procedures in this document permit the customer service personnel to have access to components that might involve mechanical hazards or pinch points, there will be instructions to shutdown, turn off, and lock out the AC input to the machine. In this manner, the customer service personnel will not be exposed to mechanical hazards or pinch points.

Moving parts on the outside of the system include the robot handler, pre-aligner, and wafer shutter door, as well as the floating optics platform. The user should take care to keep body parts and clothing clear of all moving parts while the instrument is under power.

E.3. PM Schedule:

Figure E2 represents the recommended PM intervals for the Ultrapointe Model 1000 and the responsibility for each procedure.

E.3.1 System Startup

The system is started from a power off condition by pressing the green power-on button at the front of the machine. The large red emergency off (EMO) mushroom button must be released before power will be applied; release the emergency off by rotating it counter-clockwise.

The computer operating system will go through its normal initialization and boot process, and finally the Ultrapointe application will appear on the screen. A series of initialization windows will appear; follow the instructions in the windows as they appear. Two components in the system have to be explicitly initialized: the XYZ stage and the robot. The last window to pop up as part of the initialization process is a Manual Stage Control window; click on the 'InitStage' button to initialize the stage. The robot must be initialized via the cassette window: click on 'Initialize' to initialize the robot. Neither the stage nor the robot will be operable until they are initialized.

E.3.2 System Shutdown

The system may be powered off at any time using the red 'off' button or the large EMO switch on the front panel. However, the UNIX operating system used internally on the computer platform will incur small amounts of 'damage' to its file system if the system is turned off without shutting down the computer in an orderly manner. This damage is routinely repaired as the computer system boots up at power on. If the system is repeatedly turned off without an orderly shutdown, the accumulated damage may be enough so that the automatic repair process fails. Before turning off power to the system, a system shutdown process is therefore recommended.

To perform a system shutdown, lower the Ultrapointe application: click and hold on the small square at the extreme top left of the screen. A pull-down menu appears; select 'Lower', to lower the application. After the Ultrapointe application is lowered, a set of buttons appears at the top left of the screen:

Procedure Name	Reference Paragraph	Description	Interval	Responsibility
System Startup	3.1	How to start the Ultrapointe application program	N/A	Customer
System Shutdown	3.2	Shutdown procedure for the LIS in preparation for power off	N/A	Customer
Lockup Recovery	3.3	Procedures for recovery from software lockups	N/A	Customer
Wafer Removal	3.4	Wafer removal procedures in the event of a lockup or failure	N/A	Customer
Wipe Down	3.5	Wipe down of the exterior of the machine	Daily	Customer
Backup Data	3.6	Procedure to backup critical data to tape	Weekly	Customer
Chuck Cleaning	3.7	Cleaning procedures for the chuck and robot end effector	Weekly	Customer
Laser Life and Interlock Check	3.8	Test procedure for the laser system to determine whether to schedule laser service, and to verify interlock integrity	Monthly	Customer

Robot Alignment Check	3.9	Test to verify that the robot handles wafers and interfaces appropriately to the cassette, pre-aligner and XYZ stage	Quarterly	Customer
Bulb Change	3.10	Install new halogen bulb for white light illumination	6 months	Customer
Bi-annual PM	3.11	Optics cleaning and alignment, system PM	6 months	Ultrapointe

Figure E2. Recommended PM Intervals.

Click on 'System' and another pulldown menu appears. Click on 'System Shutdown'. A window will pop up asking if you really want to shut the system down; click on 'Yes'. The computer system will now begin its shutdown process. After a minute or so, a message appears indicating that shutdown is complete and that it is OK to turn off the machine. At this point, turn off the machine by pressing the red 'Off' button or the EMO on the front panel.

E.3.3 Lockup Recovery

It is possible for the system to lock up to varying degrees. There have been reports of cases where the system slows down noticeably, of cases where none of the on-screen buttons in the Ultrapointe application respond to the cursor, and of cases where the mouse and keyboard don't seem to work at all. There is a special case of lockup that can occur during a volume acquisition. These incidents should be rare occurrences and Ultrapointe is very interested in knowing about and fixing lockups that have any frequency. The possible recovery actions include: re-starting the Ultrapointe application program, re-booting the computer system, re-starting the XWindows server (five finger reset), and cycling power of the entire instrument. Whatever the recovery method, the problem and the corrective action should be recorded for maintenance purposes (see sections 4.0 and 5.0). The following table lists lockup symptoms and recovery actions recommended. For the sake of simplicity, the operator can attempt to recover from all lockups using the reboot procedure - this takes slightly longer than just re-starting the Ultrapointe application.

Lockup Symptom	Recovery Action	Paragraph #
System slows down	Restart the application	3.3.1
Application doesn't respond	Restart the application	3.3.1
Mouse doesn't respond	Five finger reset	3.3.2
Lockup during volume acquire	Reboot the computer	3.3.3
Cannot lower the application	Five finger reset	3.3.2
Nothing works	Cycle power	3.3.4

E.3.3.1 Restarting the Ultrapointe Application

To restart the Ultrapointe application, lower the Ultrapointe application: click and hold the square at the extreme top left of the screen. A pull-down menu appears; select 'Lower' to lower the application. After the Ultrapointe application is lowered, a set of buttons appears at the top left of the screen; click on 'Tools' and another pulldown menu appears. Click on 'Shell' and after a few seconds, the outline of a UNIX shell window will appear. Position the window appropriately and click the mouse to create the window. Position the cursor anywhere in the UNIX shell window and type 'uv1000' to restart the Ultrapointe application.

If the application cannot be lowered, attempt the five finger reset as described in section 3.3.2.

E.3.3.2 Executing a Five Finger Reset

If the application cannot be lowered because the box at the extreme top left of the screen does not respond to the mouse, the operator can try to kill and restart the XWindows server using the keyboard. This involves pressing five keys simultaneously, and is referred to as the 'five finger reset'. To execute a five finger reset, press the 'Ctrl', 'Shift', 'Esc', 'F12', and '/' keys simultaneously, where the '/' key is on the numeric keypad. This should cause a dramatic change in the display, followed by a recovery to a logon screen know as the 'Pandora' screen. The Pandora screen has several icons with user names under them. Double click on the icon labeled 'uv' and the system will restart normally.

If the lockup occurred during a volume acquisition, the operator must still reboot the system even though the application has restarted. The reboot procedure is described in section 3.3.3.

If the five finger reset does not work, cycle power as described in section 3.3.4.

E.3.3.3 Rebooting the Computer

To reboot the workstation, lower the Ultrapointe application: click and hold the square at the extreme top left of the screen. A pull-down menu appears; select 'Lower' to lower the application. After the Ultrapointe application is lowered, a set of buttons appears at the top left of the screen; click on 'Tools' and another pulldown menu appears. Click on 'Shell' and after a few seconds, the outline of a UNIX shell window will appear. Position the window appropriately and click the mouse to create the window. Position the cursor anywhere in the window and type 'reboot' to reboot the computer.

If the application cannot be lowered, attempt the five finger reset as described in section 3.3.2, and then execute the reboot procedure.

E.3.3.4 Cycling Power to the Instrument

If none of the above measures is successful in returning the application to normal, the only alternative is to cycle power to the instrument. Needless to say, this will be done without a proper shutdown process. To cycle power, shut off power using the red power 'off' button on the front panel, wait 30 seconds, then power on by pressing the green 'on' button. The system should come up as described in section 3.1. Be sure that the EMO switch is released by turning counter-clockwise, or the 'on' switch will have no effect.

E.3.4 Wafer Removal

There are two conditions under which a wafer can be left in the instrument. First, the system may lose AC power or the application software may lockup while a wafer is inside the optics chamber or in the pre-aligner; the operator will therefore have to remove the wafer (using the application software) immediately after re-starting the Ultrapointe application (see section 3.4.1). Second, the instrument may become non-functional (due to a component failure) while a wafer is in the optics chamber or in the pre-aligner, and the operator may want to remove the wafer manually because the application software cannot be re-started, even by cycling power to the instrument (see section 3.4.2).

E.3.4.1 Wafer Removal Using the Application Software

Wafers can be removed from the optics chamber or the pre-aligner using the Manual Robot Control window which appears as a menu selection under the 'Maintenance' button on the menu bar across the top of the screen. This menu selection is available only to users with service or administrator permissions. However, if the Ultrapointe application is started with a wafer on the XYZ stage (in the optics chamber) or on the pre-aligner, then the Manual Robot Control menu will appear automatically when the operator attempts to initialize the robot.

With the Manual Robot Control menu on the screen, the operator clicks on the a wafer source or destination in the list at the left of the window, then clicks on 'Robot Get' or 'Robot Put' to command the robot to either retrieve or deposit a wafer at that location. **This window should be used with great care, because the usual safeguards in the applications software are not active in this window.** For example, the Manual Robot Control menu will allow you to attempt to put two wafers in the same slot, or replace a wafer into a cassette of the wrong size.

To remove a wafer from the XYZ stage, click on the 'Stage' selection in the source/destination list at the left of the window, then click on 'Robot Get'. The robot will enter the optics chamber and retrieve the wafer. To put the wafer back into the cassette, first inspect the cassette to find a slot that is empty, then click on the appropriate slot number in the source/destination list at the left of the window. Click on 'Robot Put' to deposit the wafer in that slot. **Be sure that the cassette size matches the wafer, that the slot in question is empty, and that the correct slot has been highlighted in the**

list at the left of the window before clicking on 'Robot Put'. The procedure for removing a wafer from the pre-aligner is similar, except that a 'Robot Get' operation is performed with the 'Flat Finder' selection highlighted in the source/destination list.

E.3.4.2 Manual Wafer Removal

If the Ultrapointe application software is inoperable, the user may remove a wafer from any of the major components in the system manually. The operator should be aware that he is interacting with mechanical components that move under computer control, so **it is highly recommended that the system be shut down using the procedure of section 3.2 before attempting manual wafer removal.**

If a wafer is on the robot, it can be easily removed with a vacuum wand. When the system is powered off, the operator can manually move the robot arm to a more appropriate position to remove the wafer. The default state of the vacuum hold-downs in the system is 'on', so **it is highly recommended that the operator disconnect the vacuum supply at the rear of the instrument before attempting to remove a wafer from the end effector or the chucks.** The same procedure applies to the pre-aligner module - if a wafer is left on its chuck, it can be removed manually with a vacuum wand.

If a wafer is to be removed from the XYZ stage in the optics chamber and the system is inoperable, the first step is to remove the facia plate that surrounds the wafer door. The facia plate is fastened with two screws near its base, remove these two screws and gently tilt out the bottom of the plate to remove it. Behind the facia plate is the wafer door; this can be raised manually. With the wafer door raised, reach into the system with a vacuum wand and remove the wafer from the XYZ chuck.

E.3.5 System Wipe Down

System wipe down consists of cleaning the exterior of the machine with a cleanroom-compatible cleaning cloth and a dilute solution of isopropyl alcohol or plain water. The user should check with the Ultrapointe factory if other solvents or cleaning solutions are to be used.

E.3.6 System Backup and Restore

To avoid loss of data due to a hard disk failure or other disasters, the user should periodically backup critical data. The following procedures describe backup and restore operations.

E.3.6.1 System Backup

To use the backup facilities, place a 150 megabyte quarter inch tape in the tape drive within the drive bay and close the door of the tape drive. Lower the Ultrapointe application program: click and hold on the small square at the extreme top left of the screen. A pull-down menu appears; click on 'Lower', to lower the application. A set of buttons appears at the top left corner of the screen, click on 'System', then 'System Manager'. A window appears with several icons and names above them. Double click on 'Backup&Restore'. Another window appears for setting up the backup operation; be sure that 'QIC150#7' is selected as the tape drive, and 'Backup' as the operation. Click on 'Accept' to accept the setup. Yet another window appears for specifying the files to backup. Select 'by Directory' as the files to backup, type in '/usr/uv' as the directories and files to be backed up, then click on 'Start Backup'. **Note: this is a typical backup operation, more or less data may be backed up depending on the user's preference - contact the Ultrapointe factory for recommendations.** The backup now begins, with the 'Backup is' indicating 'Active'. The backup proceeds with the list of files being backed up scrolling through the window until the backup is complete. At this point the 'Backup is' status changes to 'Inactive'.

E.3.6.2 System Restore

The restore procedure is very similar to that of section 3.6.1. Instead of selecting 'Backup' as the operation in the first window, select 'Restore'. When the setup is accepted, a window will pop up asking the operator to wait while the tape is being read. A window then appears to specify what files to restore. Be sure to specify that files should be restored to the same directory they came from, then click on 'Start Restore' to begin the restoration process. Just as before a status message indicates whether the restore process is 'Active' or 'Inactive' (complete).

E.3.7 Chuck Cleaning

The backside of wafers come into contact with three pieces of equipment in the LIS: the robot end effector, the XYZ stage chuck, and the pre-aligner chuck. These three surfaces must be cleaned periodically. Cleaning the robot end effector is straightforward using a cleanroom compatible clean wipe and a cleaning solution of either isopropyl alcohol or water. A special tool is supplied by Ultrapointe for cleaning the two chucks. It should be fitted with a fresh cleaning sleeve and used with a alcohol or water to wipe off the surface of both chucks. Do not use acetone with the chuck cleaning tool.

Cleaning the pre-aligner chuck is also straightforward. If the robot is initialized, it will move down as far as possible so that it does not interfere with the cleaning process. The chuck is visible through the slot through which the wafer enters the pre-aligner.

To clean the XYZ stage chuck, first initialize the robot to get it out of the way using the 'Initialize' button in the cassette window, select a turret position containing no objective ('00' selection from 'Objective' button at top right of LaserScan window), then open the wafer door using the LON Diagnostics window. The LON Diagnostics window is accessed by clicking on 'Maintenance' on the top menu bar, and then on 'LONDiagnostics'. A button called 'WaferDoor' appears roughly in the middle of the window; click on it to open or close the wafer door. With the wafer door open, the operator may reach in with the chuck cleaning tool to clean the surface of the chuck.

The operator should be aware that he is interacting with mechanical components that move under computer control, so **it is highly recommended that the system be shut down using the procedure of section 3.2 before attempting chuck cleaning.**

Close the wafer door using the LON Diagnostics window when the cleaning operation is complete.

E.3.8 Laser Lifetime and Interlock Inspection

The normal aging process of the laser tube can be monitored by measuring the current required to obtain a given power output. The procedure involves setting the laser power to maximum (25 milliwatts) and measuring the required tube current. With a new laser this is typically 6 Amps. The laser power supply is capable of supplying a maximum of 10.5 amperes. When the laser tube requires 9.5 amps to achieve 25 milliwatts of power it is close to the end of its life (est. 1000 hours remaining) and a replacement should be scheduled.

The procedure is as follows:

1) Click on the 'Maintenance' selection on the menu bar at the top of the screen. Select 'LONDiagnostics' from the resultant pull-down menu.
2) On the right hand side of the LON Diagnostics screen are two columnar lists: the left column (labeled 'Nodes') lists the I/O boards in the instrument, the right column (labeled NetVariables) lists the available network variables for the I/O board selected in the left column. Select the 'LaserPwrControl' in the left column.
3) Select the 'PowerCmd' network variable in the right column.
4) Click on the text entry box below the left column (labeled OutputNVValue) and enter 128, which corresponds to 25 milliwatts of laser power. Click on the 'SendToLON' button to change the current laser power setting to 25mW.
5) Click on the 'CurrMonitor' network variable in the right column.
6) Check and record the value returned as 'NetVarValue:' below the left column.

7) Divide the returned value recorded in step 6 by the conversion factor of 2.3052. This gives the current in amperes required to deliver the 25 milliwatts of laser power. Record this value.
8) If the current required to deliver 25 milliwatts of laser power is greater than 9.5 Amps, the laser is nearing the end of its life. A factory service call should be scheduled to replace the laser.

The wafer door interlock should be tested at the same time as the laser lifetime is inspected. To test the interlocks, load a wafer of any kind into the instrument using routine operating procedures. Focus on the surface of the wafer and obtain a laser image on the workstation screen. The user then opens the wafer door using the LON Diagnostics menu and verifies that the laser image goes away because the wafer door interlock has shuttered the laser. The LON Diagnostics menu can only be accessed by users with service or administrator permissions. It is reached by clicking on 'Maintenance' on the menu bar at the top of the screen, then clicking on 'LONDiagnostics' on the resultant pull-down menu. In the LON Diagnostics menu is a button labeled 'WaferDoor'; click on this button to open or close the wafer door. Verify that the laser image disappears when the door is open and re-appears when the door closes.

E.3.9 Robot Alignment Check

Testing robot alignment is a qualitative procedure to ensure that the robot is interfacing properly to the cassette, pre-aligner, and XYZ stage. The only reason that robot alignment should change is if one of the components in question suffers a mechanical impact of some kind.

E.3.9.1 Robot to Cassette Alignment Check

Bring up the Manual Robot Control window by clicking on 'Maintenance' on the top menu bar, and then on 'ManualRobotControl'. With a full cassette of wafers on the cassette platform, highlight each of the cassette slots in the source/destination list and use the 'Robot Get' and 'Robot Put' buttons to load and replace wafers from all 25 slots. **Use extreme care with the Manual Robot Control window because the safeguards in the normal operating software are not in effect when using this window:** for example, the system will not prevent the user from attempting to put two wafers in the same slot from the Manual Robot Control window.
As the robot's end effector goes in and out of the cassette to pick up and replace each wafer, observe that the wafers are moved smoothly and that there is no rubbing between the wafer and the cassette.

E.3.9.2 Robot to Pre-Aligner Alignment Check

Using the Manual Robot Control window in the same manner as described in section 3.9.1, execute a 'Robot Get' operation to retrieve a wafer from the cassette, then execute a 'Robot Put' with the pre-aligner highlighted in the source/destination list. Observe the motion of the robot as it enters the pre-aligner and be sure that there is no rubbing between the robot and any parts of the pre-aligner. Check that the robot places the wafer gently on the pre-aligner chuck and picks it back up in the same manner.

E.3.9.3 Robot to XYZ Stage Alignment Check

This operation proceeds in exactly the same manner as section 3.9.2, except that the 'Stage' selection is highlighted in the source/destination list. As the robot enters the optics chamber to drop off or retrieve a wafer, be sure that there is no rubbing between the robot or wafer and the wafer door facia plate or the chuck on the XYZ stage.

E.3.10 Changing the Halogen Bulb

Changing the halogen bulb requires taking off the top cover of the machine and manipulating the lamp house on the optics plate. The top cover is interlocked with the laser power supply and should cause the laser to shut off when the cover is removed. It cannot be turned back on until the interlocks are closed and power is cycled. **The top cover interlocks are defeatable, but under no circumstances are customer service personnel authorized to defeat the laser interlocks. If at any time, the customer service personnel are aware of laser radiation (intense blue-green light) while the top cover of the instrument is removed, the instrument should be immediately powered off and a service call placed to the Ultrapointe factory.**

With the top cover removed, the customer service personnel can have access to components that are at a high voltage. There is no danger of contacting high voltage components unless the operator disassembles components that have nothing to do with the procedure at hand. **Use of controls or adjustments, or performance of procedures other than those specified here may result in the customer service personnel being exposed to unnecessary electrical, mechanical, or radiation hazards.**

The halogen bulb used for white light illumination gets extremely hot during normal operation. **Take care before handling the bulb or the lamp house that sufficient time has been allowed after powering off the system to allow the lamp components to cool.**

The procedure for replacing the halogen bulb is as follows:

1) Turn off the machine using the shutdown procedure described in section 3.2.
2) Release the top cover of the machine by unscrewing the four quarter-turn captive screws, one at each corner of the top cover. Carefully lift off the cover and place it aside in a clean and safe place.
3) Wait fifteen minutes for the lamp house to cool down.
4) Identify the white light lamp house (see figure 2): it is a roughly rectangular housing, off-white in color, with black louvers facing up to allow heat to escape. The lamp house is on the optics plate immediately beneath the top cover. The lamp socket is a cylindrical object that fits into the side of the lamp house and carries the power cord for the lamp.
5) There are two thumbscrews attached to the lamp socket. One of them protrudes radially from the socket and is used to adjust and lock the angular position of the lamp. The other protrudes axially from the socket and is used to adjust the axial position of the lamp. Loosen the radial thumbscrew and observe that the socket can now be rotated within the lamp house.
6) Rotate the lamp house clockwise until it contacts a stop, then pull the socket out axially.

7) The bulb is held in its socket by gold contacts. Press down on the long lever arms of the gold contacts to release the bulb; pull the bulb out of its socket while the levers are depressed.

8) Install a new bulb (generic part number: FCR 12V, Ultrapointe part number 000616) by depressing the socket levers while pressing the bulb into the socket. Take care not to touch the bulb because any contamination on the outside of the bulb will lead to premature failure.

9) Replace the socket and bulb in the lamp house by reversing the process of step 5.

10) Snug down the radial thumbscrew to hold the socket in place and turn on the instrument using the procedure of section 3.1.

11) Load a patterned wafer into the machine using normal operating procedures. The autofocus operations will not work because the laser interlocks are open.

12) Manually focus on the wafer using the joystick and the lowest power objective available.

13) Loosen the radial thumbscrew on the lamp socket and rotate the lamp socket to obtain the most uniform illumination of the sample, as indicated by the white light image of the wafer. Snug down the radial thumbscrew.

14) Adjust the axial thumbscrew to obtain the most uniform illumination of the sample. These adjustments are moving the position of the lamp filament with respect to the optics train.

15) Repeat steps 13 and 14 until the illumination is optimized. Tighten the radial thumbscrew.

16) Replace the top cover of the machine and screw in the four quarter turn fasteners that hold it to the machine frame. This operation closes the laser interlocks.

17) Turn off the machine as described in section 3.2, then restart it as described in section 3.2. Cycling power should bring back the laser back up.

E.3.11 Ultrapointe Factory PM

Ultrapointe factory PM procedures are to be performed only by factory authorized personnel. The procedures include cleaning and re-alignment of the optics train, which is a sensitive process that is critical to the performance of the instrument. The procedures involved are described separately in internal Ultrapointe documents.

E.4. Preventative Maintenance Log Sheet

The following form should be used to log regular PM procedures.

Date	Procedure Description	Initiator	Comment

E.5. Problem Report Worksheet

Use the following form to report bugs and problems to the Ultrapointe factory. Fill in as much as possible in the sections that are not shaded. Mail or fax the worksheet to Ultrapointe at:

Ultrapointe Corp.
163 Baypointe Pkwy
San Jose, CA 95134
tel: 408 894 7080
fax: 408 894 7090
Attn: Bug List Mgr.

Date:			
Reported by:		**Company:**	
Description of problem:			
Operating Conditions:			
Software Revision/Date:			
Screen in use:			
How did you recover?			
Recommendation:			
Assigned to:		**Date:**	

What is claimed is:

1. A method for automatic focusing of a confocal microscope, said microscope transmitting a laser beam through a lens and through a pinhole of a spatial filter to a surface of a target, said method comprising the steps of:

moving a target relative to a lens of said microscope, said target being moved in a predetermined first direction through a predetermined first distance;

generating an electronic focus signal during movement of said target, the magnitude of said electronic focus signal being a function of the magnitude of light from said laser beam that is reflected from said surface and passes through said pinhole;

recording a plurality of first values of the magnitude of said electronic focus signal during the movement of said target in said first direction; and

stopping said movement of said target when said target has travelled through said first predetermined distance, said first predetermined distance being greater than the depth of focus of said lens, said depth of focus being larger than the distance between two adjacent positions at which said first values are recorded, wherein after stopping, said target is at a position other than a position at which said microscope is focused.

2. The method of Claim 1, further comprising, after said stopping step the steps of:

calculating a first estimate of a focus position at which the microscope is focused by using said plurality of first values;

moving said target relative to said lens in a second direction, said second direction being opposite said first direction;

generating said electronic focus signal during movement of said target in said second direction;

recording a plurality of second values of said magnitude of said electronic focus signal during movement of said target in said second direction; and

stopping said movement of said target in said second direction when said target reaches a stop position, wherein said stop position is derived from said first estimate.

3. The method of Claim 2 further comprising:

calculating a second estimate of the position at which said microscope is focused by using said plurality of second values; and

moving said target to said second estimate.

4. The method of Claim 2 wherein said calculating step comprises:

summing all of said plurality of first values to generate a first sum;

dividing said full-sum by 2 to generate a half-sum; and

repeatedly summing a sequence of said plurality of first values to generate a plurality of partial sums until one of said plurality of partial sums exceeds said half-sum.

5. The method of Claim 4 further comprising using the elevation at which said partial sum exceeds said half-sum as said first estimate.

6. The method of Claim 4 further comprising determining a starting position and a stopping position, said stopping position and said starting position being on opposite sides of an estimated focus position, wherein said starting position is at a distance of at least a depth of focus of said lens from said estimated focus position and further wherein said stopping position is at said distance.

7. The method of Claim 6 further comprising a start-up move for moving said target to said starting position if said target is away from said starting position.

8. The method of Claim 2, wherein said target is moved through said first distance at a first velocity and through said second distance at a second velocity, said second velocity being less than said first velocity.

9. The method of Claim 1, further comprising calculating a first estimate of the position at which the microscope is focused by using said plurality of first digitized values.

10. The method of Claim 9, further comprising moving said target to said first estimate.

11. The method of Claim 1, wherein said target is moved by a stepper motor or by a piezo electrically driven element.

12. The method of Claim 1, wherein said step of generating further comprises:

measuring said intensity of light reflected from said surface of said target through said pin hole in a direction reverse to the incident path of said laser beam; and

transforming said measured intensity into said electronic focus signal.

13. The method of Claim 12, wherein said step of transmitting further comprises moving said laser beam to define a line on said surface of said target.

14. The method of Claim 1 further comprising the step of setting up gain and adjusting the gain if said target moves through said first distance without said absolute value of said magnitude of said electronic focus signal being a predetermined optimal value.

15. The method of Claim 1 wherein said recording occurs at periodic intervals.

16. The method of Claim 1 wherein the majority of said first digitized values are non-zero.

17. A method for positioning a target close to a focus position in a confocal microscope, said confocal microscope generating an electronic focus signal having a magnitude proportional to light reflected by said target, said electronic focus signal having a peak that indicates a focus condition, said method comprising the steps of:

positioning said target such that said focus position is within a first range of motion of said target;

moving said target a plurality of first steps through said first range of motion of said target, each of said first steps having a first width;

measuring a plurality of first strengths of said electronic focus signal at each of said first steps;

calculating a first estimate of the position at which said confocal microscope is focused by using said plurality of first strengths;

moving said target a plurality of second steps through a second range of motion, wherein said second range of motion includes second steps above and below said first estimate, each of said second steps having a second width;

measuring a plurality of second strengths of said electronic focus signal at each of said second steps;

calculating a second estimate of the position at which said confocal microscope is focused by using said plurality of second strengths;

moving said target to said second estimate.

18. The method of Claim 17, wherein said second range of motion is shorter than said first range of motion.

19. The method of Claim 17, wherein said second width is shorter than said first width.

20. The method of Claim 17, wherein said first width is smaller than width of said focus signal.

21. The method of Claim 17, wherein said target is moved in a first direction through said plurality of first steps and in a second direction through said plurality of second steps, said second direction being identical to said first direction.

22. The method of Claim 17, wherein said target is moved in a second direction through said plurality of second steps and in a second direction through said

plurality of second steps, said second direction being opposite to said first direction.

23. A method for automatically focusing a microscope that transmits a laser beam through a lens and through a pinhole of a spatial filter to a surface of a target, said method comprising the steps of:

moving a target relative to a lens of said microscope in a first direction;

generating an electronic focus signal for a plurality of points in an area of said target at least three points of said plurality of points defining a plane;

recording as a first value the largest value of said electronic focus signal for said plurality of points for each position in a plurality of first positions of said target with respect to said lens; and

stopping said movement of said target when said target has travelled through a first predetermined distance, wherein said first predetermined distance is at a position other than a position at which said microscope is focused.

24. The method of Claim 23, further comprising, after said stopping step the steps of:

calculating a first estimate of the position at which the microscope is focused by using a plurality of said first values;

moving said target relative to said lens in a second direction, said second direction being opposite said first direction;

generating said electronic focus signal for a plurality of points in said area of said target, at least three points of said plurality of points defining a plane;

recording as a second value, the largest value of said electronic focus signal for said plurality of points for each position in a plurality of second positions of said target with respect to said lens; and

stopping said movement of said target in said second direction when said target reaches a stop position, wherein said stop position is derived from said first estimate.

25. The method of Claim 23 further comprising:

calculating a second estimate of the position at which said microscope is focused by using said plurality of second values; and

moving said target to said second estimate.

26. The method of Claim 23 wherein said target is moved through said first distance at a first velocity and through said second distance at a second velocity, said second velocity being less than said first velocity.

27. The method of Claim 23 wherein said second direction is identical to said first direction.

28. The method of Claim 23 wherein said second direction is opposite to said first direction.

29. An apparatus for automatic focusing of a confocal microscope, said microscope having a laser, a lens and a spatial filter, said apparatus comprising:

means for moving a target relative to said lens of said microscope, said target being moved in a predetermined first direction through a predetermined first distance;

means for generating an electronic focus signal during movement of said target, the magnitude of said electronic focus signal being a function of the magnitude of light from said laser beam that is reflected from said surface and passes through said spatial filter;

means for recording a plurality of first values of the magnitude of said electronic focus signal during the movement of said target in said first direction, said means for recording being coupled to said means for generating; and

means for stopping said movement of said target when said target has travelled through said first predetermined distance, said first predetermined distance being greater than the depth of focus of said lens, said depth of focus being larger than the distance between two adjacent positions at which said first values are recorded, wherein after stopping, said target is at a position other than a position at which said microscope is focused.

30. The apparatus of Claim 29 further comprising a peak detector coupled to said means for recording, wherein said peak detector supplies the maximum value of said electronic focus signal that is encountered while a laser beam is scanned across an area of said target, while said target is held stationary.

31. A method for automatically focusing a microscope, comprising the steps of:

continuously moving a target relative to a lens of the microscope, the target being moved in a first direction through a first distance;

continuously generating an electronic focus signal during movement of the target, the magnitude of the electronic focus signal being a function of the magnitude of light reflected from a surface of the target through an optical path of the microscope;

continuously comparing the absolute value of the magnitude of the electronic focus signal to a threshold value; and

stopping the movement of the target when the absolute value of the magnitude of the electronic focus signal exceeds the threshold value;

wherein the step of continuously generating further comprises:

transmitting a laser beam through the lens to a surface of the target;

measuring the intensity of the light reflected from the surface of the target through the optical path of the microscope; and

transforming the measured intensity into the electronic focus signal;

wherein the step of transmitting further comprises moving the laser beam to define an area on the surface of the target.

AMENDED CLAIMS
[received by the International Bureau on 26 June 1995 (26.06.95);
original claims 1,17,23 and 29 amended; remaining claims
unchanged (8 pages)]

1. A method for automatic focusing of a confocal microscope, said microscope transmitting a laser beam through a lens and through a pinhole of a spatial filter to a surface of a target, said method comprising the steps of:

moving a target relative to a lens of said microscope, said target being moved in a predetermined first direction through a predetermined first distance;

generating an electronic focus signal during movement of said target, the magnitude of said electronic focus signal being a function of the magnitude of light from said laser beam that is reflected from said surface and passes through said pinhole;

recording a plurality of first values of the magnitude of said electronic focus signal during the movement of said target in said first direction; and

stopping said movement of said target when said target has travelled through said first predetermined distance, said first predetermined distance being greater than the depth of focus of said lens, said depth of focus being larger than the distance between two adjacent positions at which said first values are recorded, wherein after stopping, said target is at a position other than a position at which said microscope is focused;

wherein the step of generating comprises:

transmitting a laser beam through the lens to a surface of the target;

measuring the intensity of the light reflected from the surface of the target through the optical path of the microscope; and transforming the measured intensity into the electronic focus signal;

wherein the step of transmitting further comprises moving the laser beam to define an area on the surface of the target.

2. The method of Claim 1, further comprising, after said stopping step the steps of:

calculating a first estimate of a focus position at which the microscope is focused by using said plurality of first values;

steps;

calculating a first estimate of the position at which said confocal microscope is focused by using said plurality of first strengths;

moving said target a plurality of second steps through a second range of motion, wherein said second range of motion includes second steps above and below said first estimate, each of said second steps having a second width;

measuring a plurality of second strengths of said electronic focus signal at each of said second steps;

calculating a second estimate of the position at which said confocal microscope is focused by using said plurality of second strengths; and

moving said target to said second estimate;

wherein the step of generating comprises:

transmitting a laser beam through the lens to a surface of the target;

measuring the intensity of the light reflected from the surface of the target through the optical path of the microscope; and

transforming the measured intensity into the electronic focus signal;

wherein the step of transmitting further comprises moving the laser beam to define an area on the surface of the target.

18. The method of Claim 17, wherein said second range of motion is shorter than said first range of motion.

19. The method of Claim 17, wherein said second width is shorter than said first width.

20. The method of Claim 17, wherein said first

width is smaller than width of said focus signal.

21. The method of Claim 17, wherein said target is moved in a first direction through said plurality of first steps and in a second direction through said plurality of second steps, said second direction being identical to said first direction.

22. The method of Claim 17, wherein said target is moved in a second direction through said plurality of second steps and in a second direction through said

plurality of second steps, said second direction being opposite to said first direction.

23. A method for automatically focusing a microscope that transmits a laser beam through a lens and through a pinhole of a spatial filter to a surface of a target, said method comprising the steps of:

moving a target relative to a lens of said microscope in a first direction;

generating an electronic focus signal for a plurality of points in an area of said target at least three points of said plurality of points defining a plane;

recording as a first value the largest value of said electronic focus signal for said plurality of points for each position in a plurality of first positions of said target with respect to said lens; and

stopping said movement of said target when said target has travelled through a first predetermined distance, wherein said first predetermined distance is at a position other than a position at which said microscope is focused;

wherein the step of generating comprises:

transmitting a laser beam through the lens to a surface of the target;

measuring the intensity of the light reflected from the surface of the target through the optical path of the microscope; and

transforming the measured intensity into the electronic focus signal;

wherein the step of transmitting further comprises moving the laser beam to define an area on the surface of the target.

24. The method of Claim 23, further comprising,

after said stopping step the steps of:

calculating a first estimate of the position at which the microscope is focused by using a plurality of said first values;

moving said target relative to said lens in a second direction, said second direction being opposite said first direction;

generating said electronic focus signal for a plurality of points in said area of said target, at least three points of said plurality of points defining a plane;

signal during movement of said target, the magnitude of said electronic focus signal being a function of the magnitude of light from said laser beam that is reflected from said surface and passes through said spatial filter;

means for recording a plurality of first values of the magnitude of said electronic focus signal during the movement of said target in said first direction, said means for recording being coupled to said means for generating; and

means for stopping said movement of said target when said target has travelled through said first predetermined distance, said first predetermined distance being greater than the depth of focus of said lens, said depth of focus being larger than the distance between two adjacent positions at which said first values are recorded, wherein after stopping, said target is at a position other than a position at which said microscope is focused;

wherein the means for generating comprises:

means for transmitting a laser beam through the lens to a surface of the target;

means for measuring the intensity of the light reflected from the surface of the target through the optical path of the microscope; and

means for transforming the measured intensity into the electronic focus signal;

wherein the means for transmitting further comprises means for moving the laser beam to define an area on the surface of the target.

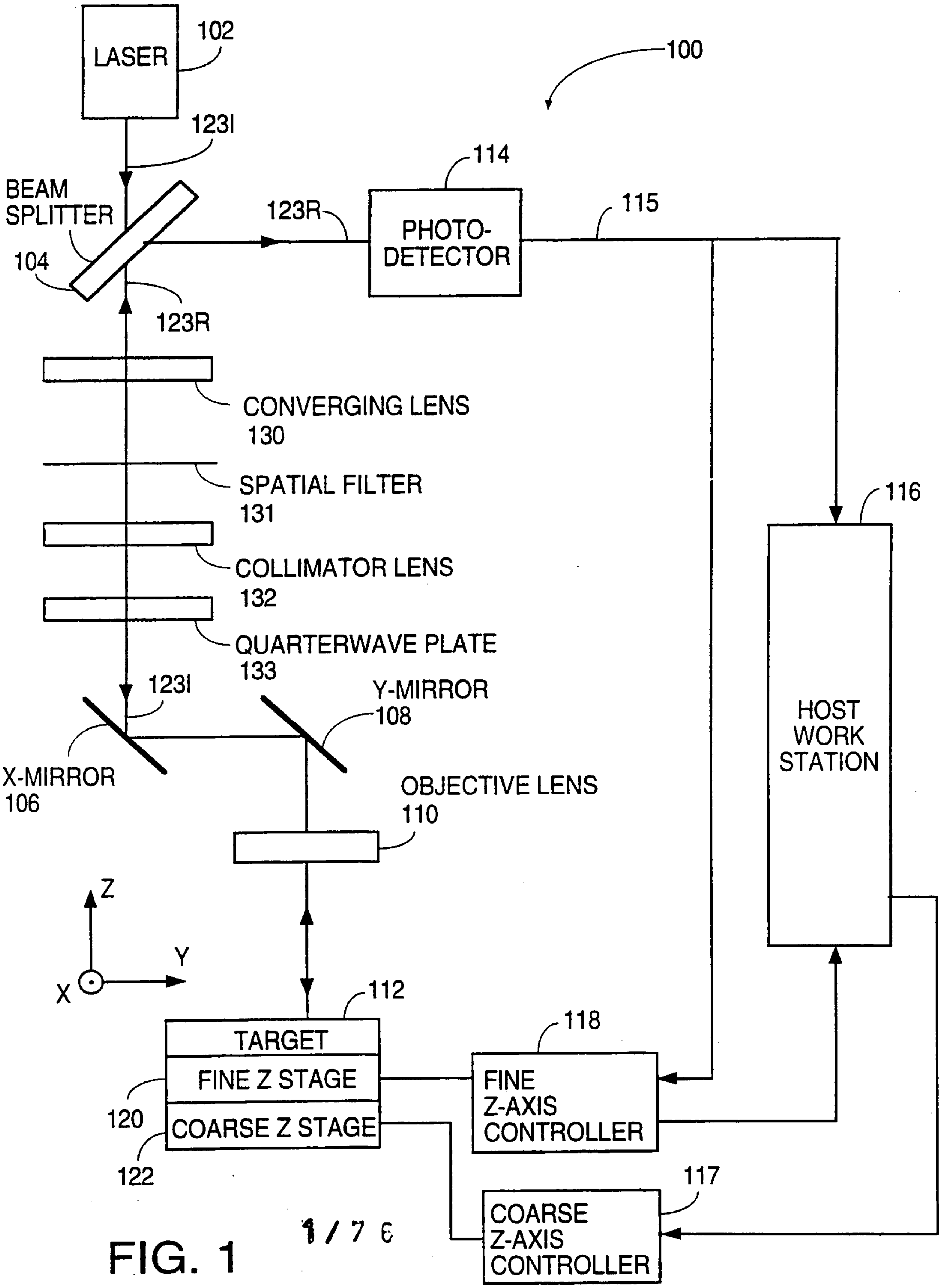
30. The apparatus of Claim 29 further comprising a peak detector coupled to said means for recording, wherein said peak detector supplies the maximum value

of said electronic focus signal that is encountered while a laser beam is scanned across an area of said target, while said target is held stationary.

31. A method for automatically focusing a microscope, comprising the steps of:

continuously moving a target relative to a lens of the microscope, the target being moved in a first direction through a first distance;

continuously generating an electronic focus signal during movement of the target, the magnitude of the electronic focus signal being a


LASER
102
100
123I
114
BEAM SPLITTER
123R
PHOTO-DETECTOR
115
104
123R
CONVERGING LENS 130
SPATIAL FILTER 131
116
COLLIMATOR LENS 132
QUARTERWAVE PLATE 133
123I
Y-MIRROR 108
HOST WORK STATION
X-MIRROR 106
OBJECTIVE LENS 110
Z
Y
X
112
118
TARGET
FINE Z STAGE
FINE Z-AXIS CONTROLLER
120
COARSE Z STAGE
122
117
COARSE Z-AXIS CONTROLLER

1 / 76

FIG. 1

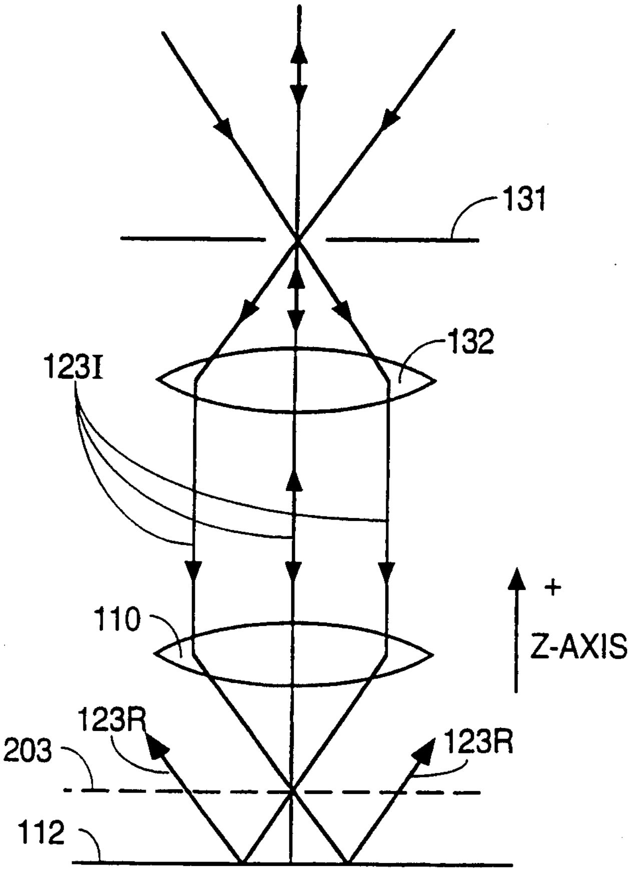

131
132
123I
110
+
Z-AXIS
123R
203
123R
112

FIG. 2A

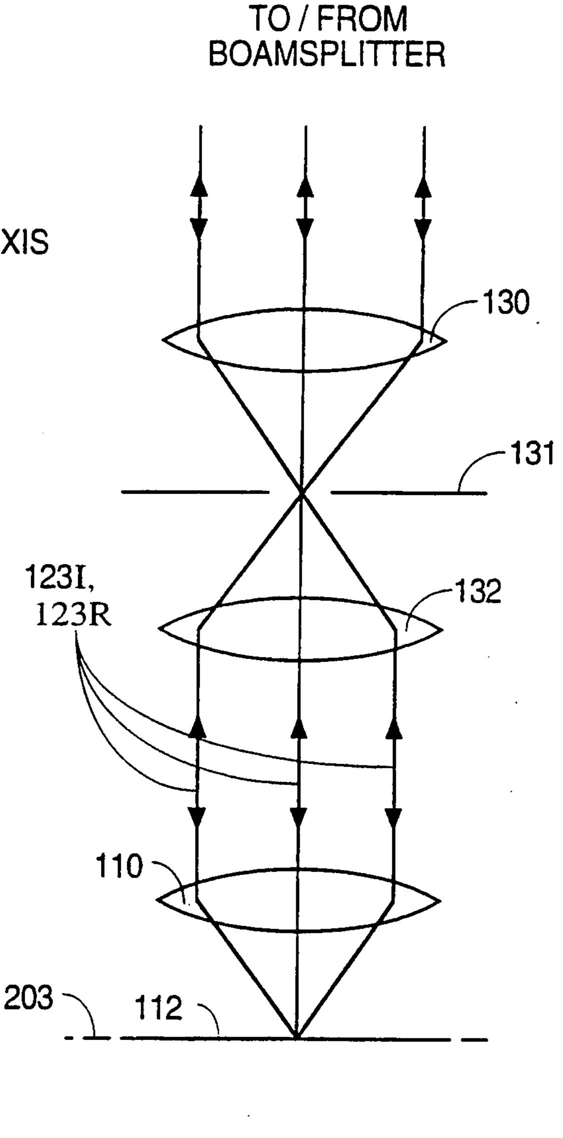

TO / FROM
BOAMSPLITTER
123I,
123R

FIG. 2B

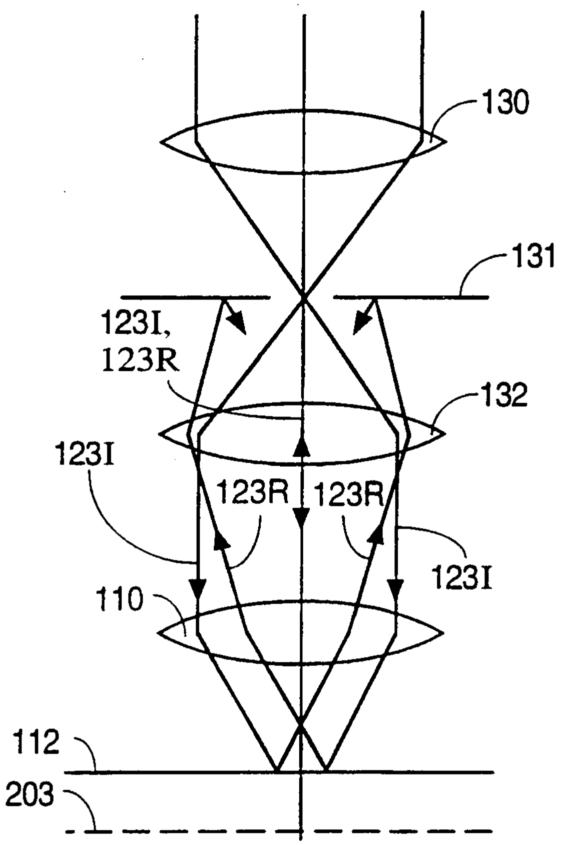

130
131
123I,
123R
132
123I
123R
123R
123I
110
112
203

FIG. 2C

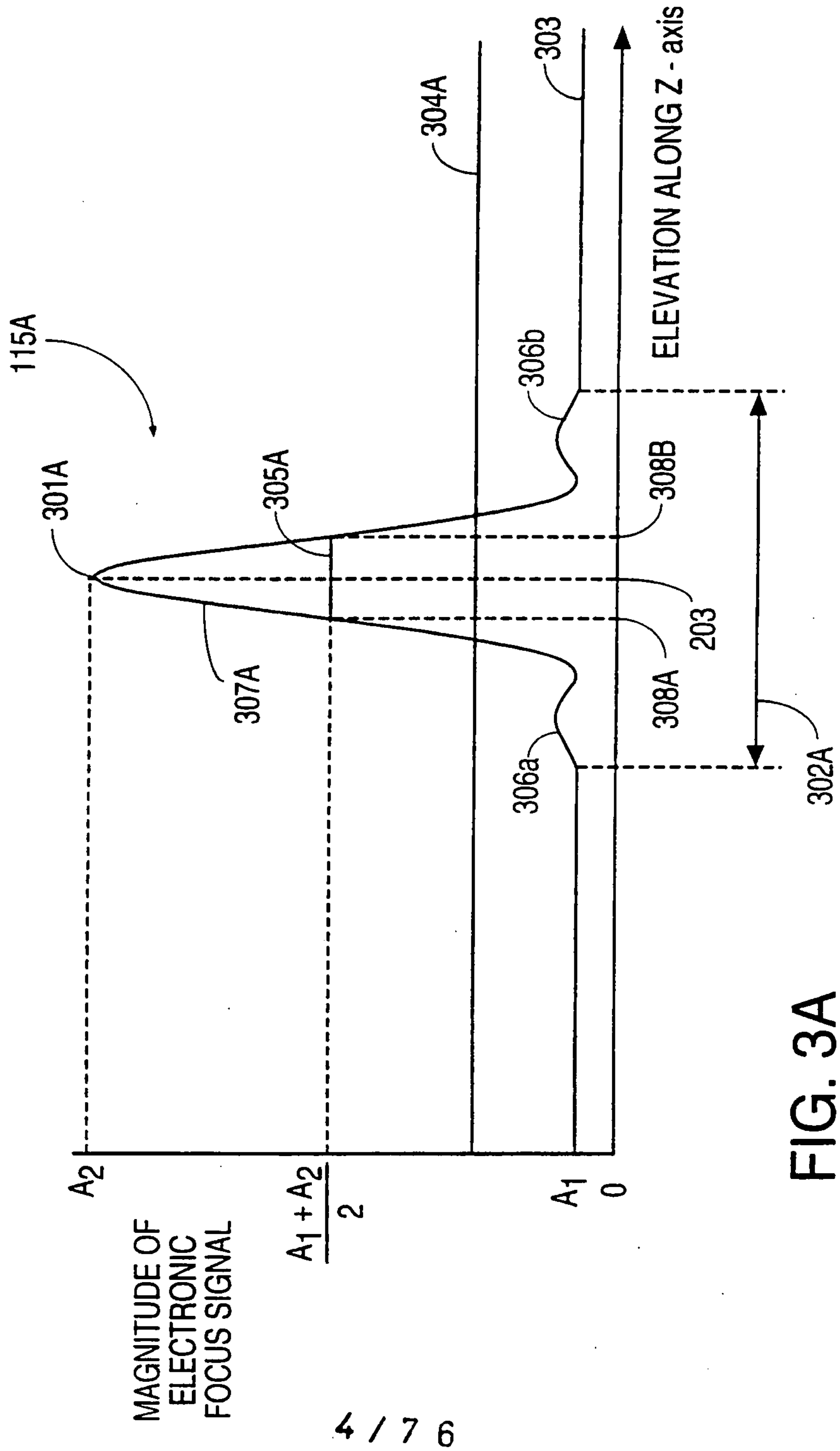

115A
301A
305A
307A
308B
203
308A
306a
306b
302A
303
304A
ELEVATION ALONG Z - axis
A_2
$\frac{A_1 + A_2}{2}$
A_1
0
MAGNITUDE OF ELECTRONIC FOCUS SIGNAL

FIG. 3A

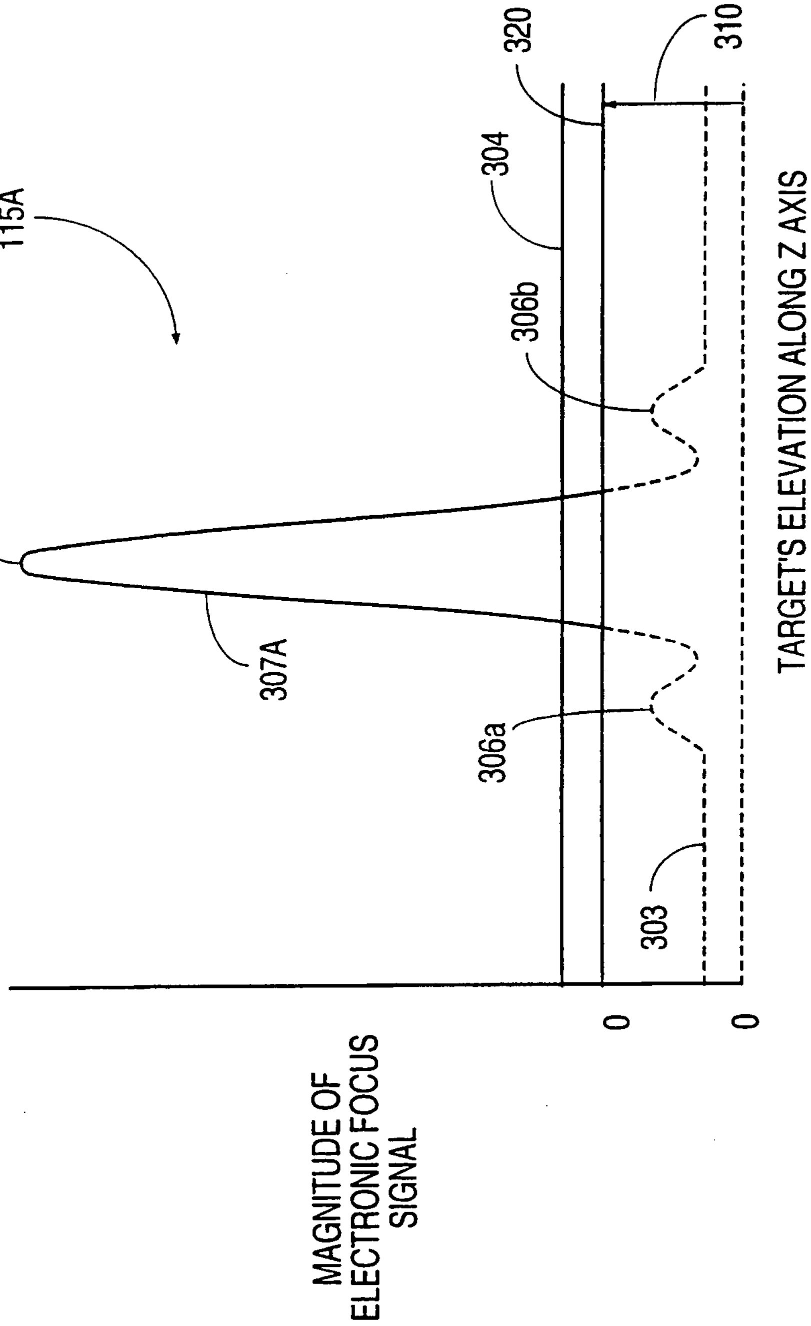

MAGNITUDE OF ELECTRONIC FOCUS SIGNAL
0
0
115A
301B
307A
306a
306b
303
304
320
310
TARGET'S ELEVATION ALONG Z AXIS

FIG. 3B

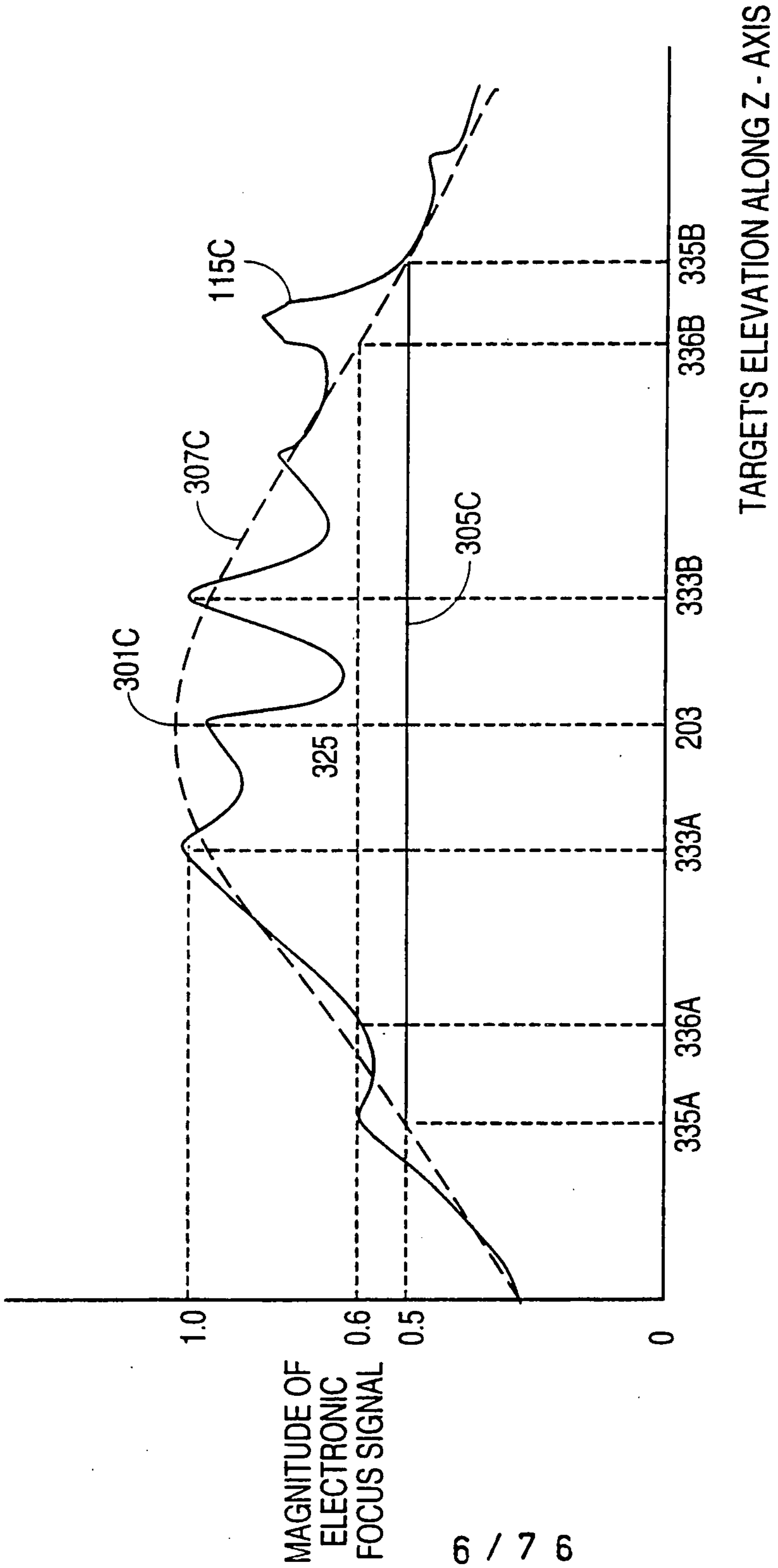

MAGNITUDE OF ELECTRONIC FOCUS SIGNAL
1.0
0.6
0.5
0
TARGET'S ELEVATION ALONG Z - AXIS
335A
336A
333A
203
333B
336B
335B
301C
307C
305C
325
115C

FIG. 3C

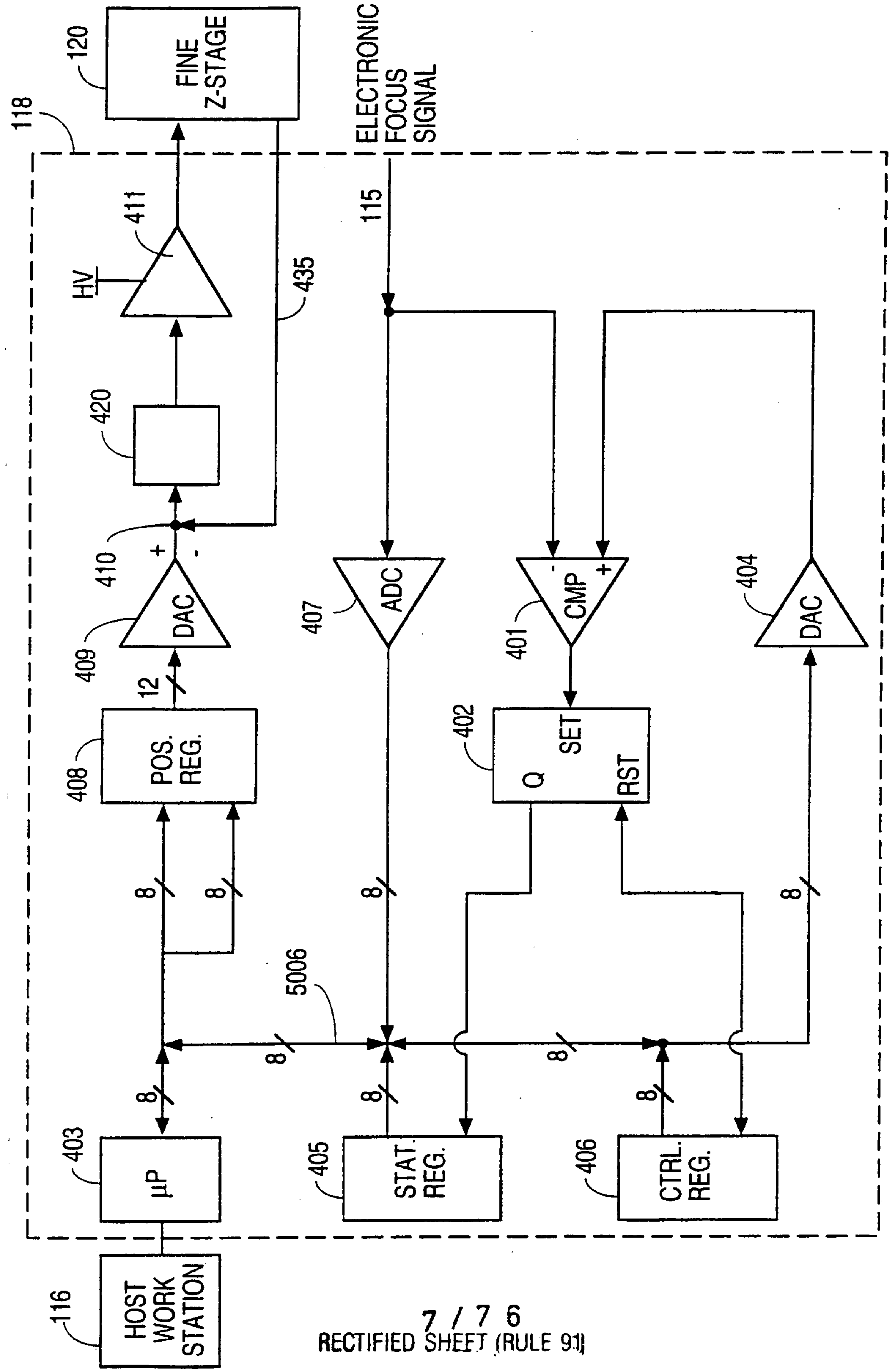

120
FINE Z-STAGE
118
ELECTRONIC FOCUS SIGNAL
115
HV
411
435
420
410
+
-
DAC
409
12
POS. REG.
408
ADC
407
CMP
401
402
SET
Q
RST
404
DAC
8
5006
403
µP
405
STAT. REG.
406
CTRL. REG.
116
HOST WORK STATION

FIG. 4

FIG 5A-1

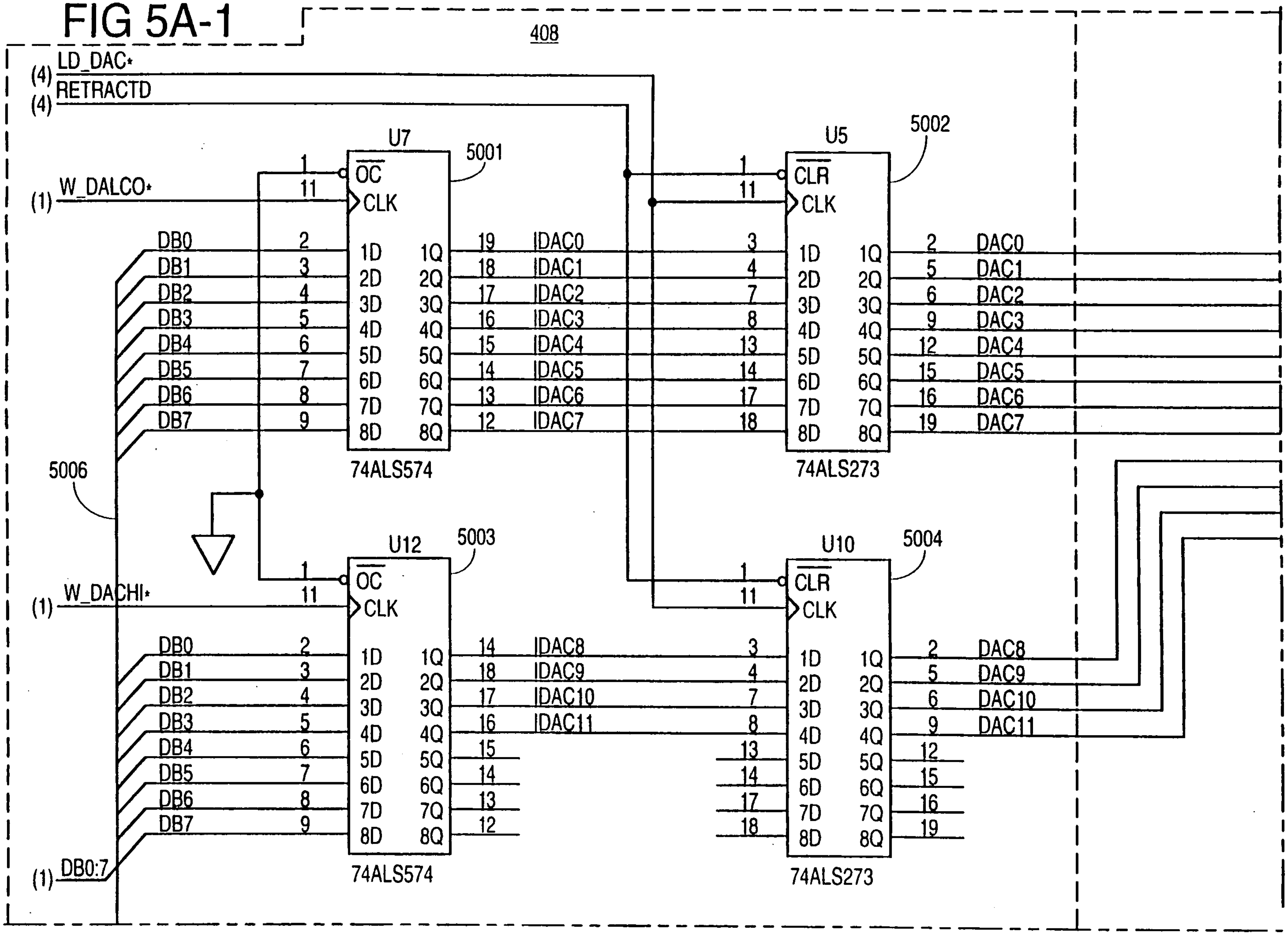
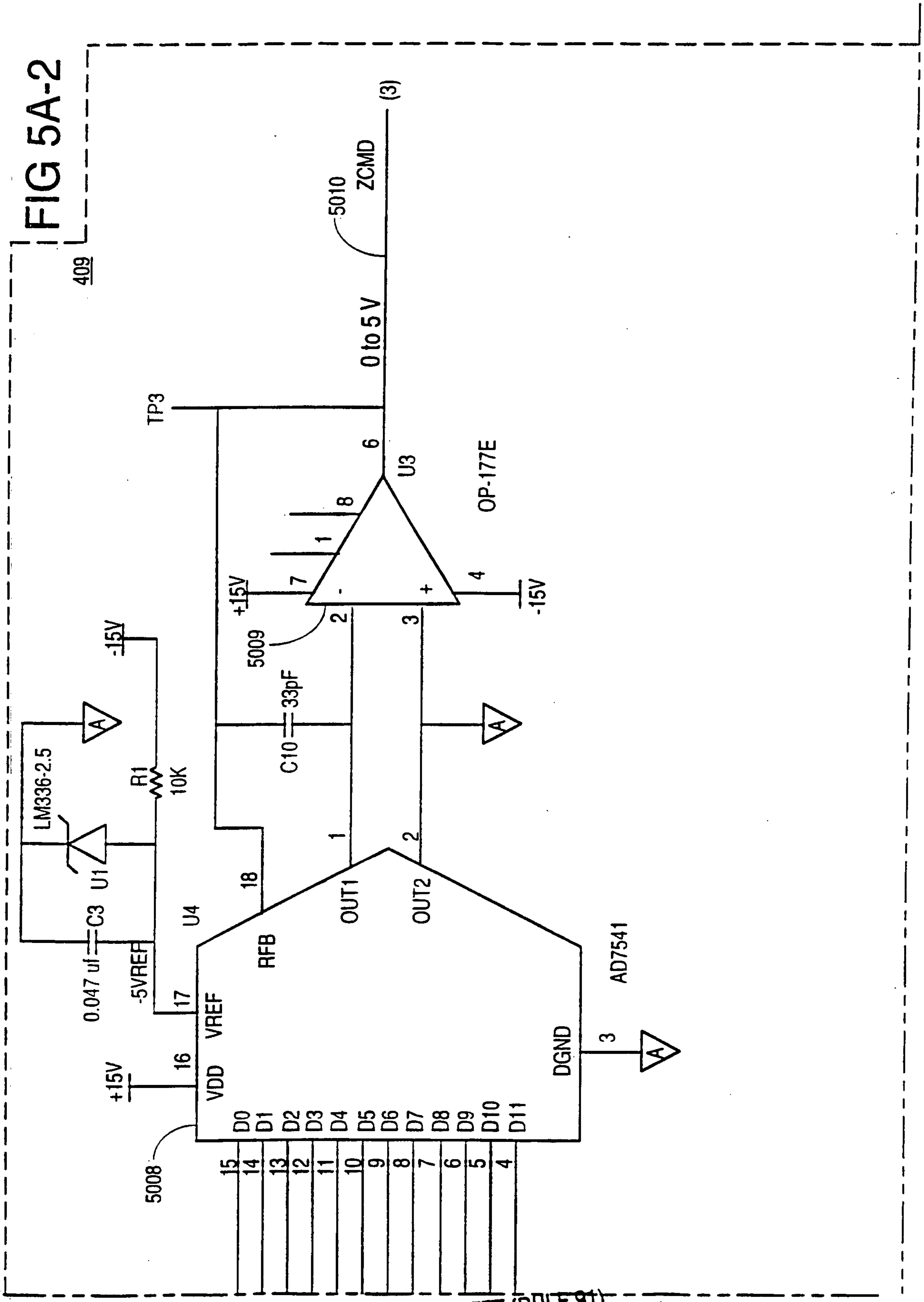

408
(4) LD_DAC*
(4) RETRACTD
(1) W_DALCO*
(1) W_DACHI*
(1) DB0:7
5006
U7
5001
OC
CLK
1D 2D 3D 4D 5D 6D 7D 8D
1Q 2Q 3Q 4Q 5Q 6Q 7Q 8Q
74ALS574
U12
5003
74ALS574
U5
5002
CLR
74ALS273
U10
5004
74ALS273
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7
IDAC0 IDAC1 IDAC2 IDAC3 IDAC4 IDAC5 IDAC6 IDAC7
IDAC8 IDAC9 IDAC10 IDAC11
DAC0 DAC1 DAC2 DAC3 DAC4 DAC5 DAC6 DAC7
DAC8 DAC9 DAC10 DAC11
1 11 2 3 4 5 6 7 8 9
19 18 17 16 15 14 13 12
14 18 17 16 15 14 13 12
3 4 7 8 13 14 17 18
2 5 6 9 12 15 16 19

FIG 5A-2


409
(3)
ZCMD
5010
0 to 5 V
TP3
6
U3
OP-177E
8
1
7
4
+15V
-15V
-
+
2
3
5009
33pF
C10
-15V
A
A
A
LM336-2.5
U1
R1
10K
C3
0.047 uf
-5VREF
U4
18
RFB
1
OUT1
2
OUT2
AD7541
17
VREF
16
VDD
+15V
DGND
3
5008
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
15
14
13
12
11
10
9
8
7
6
5
4

RECTIFIED SHEET (RULE 91)

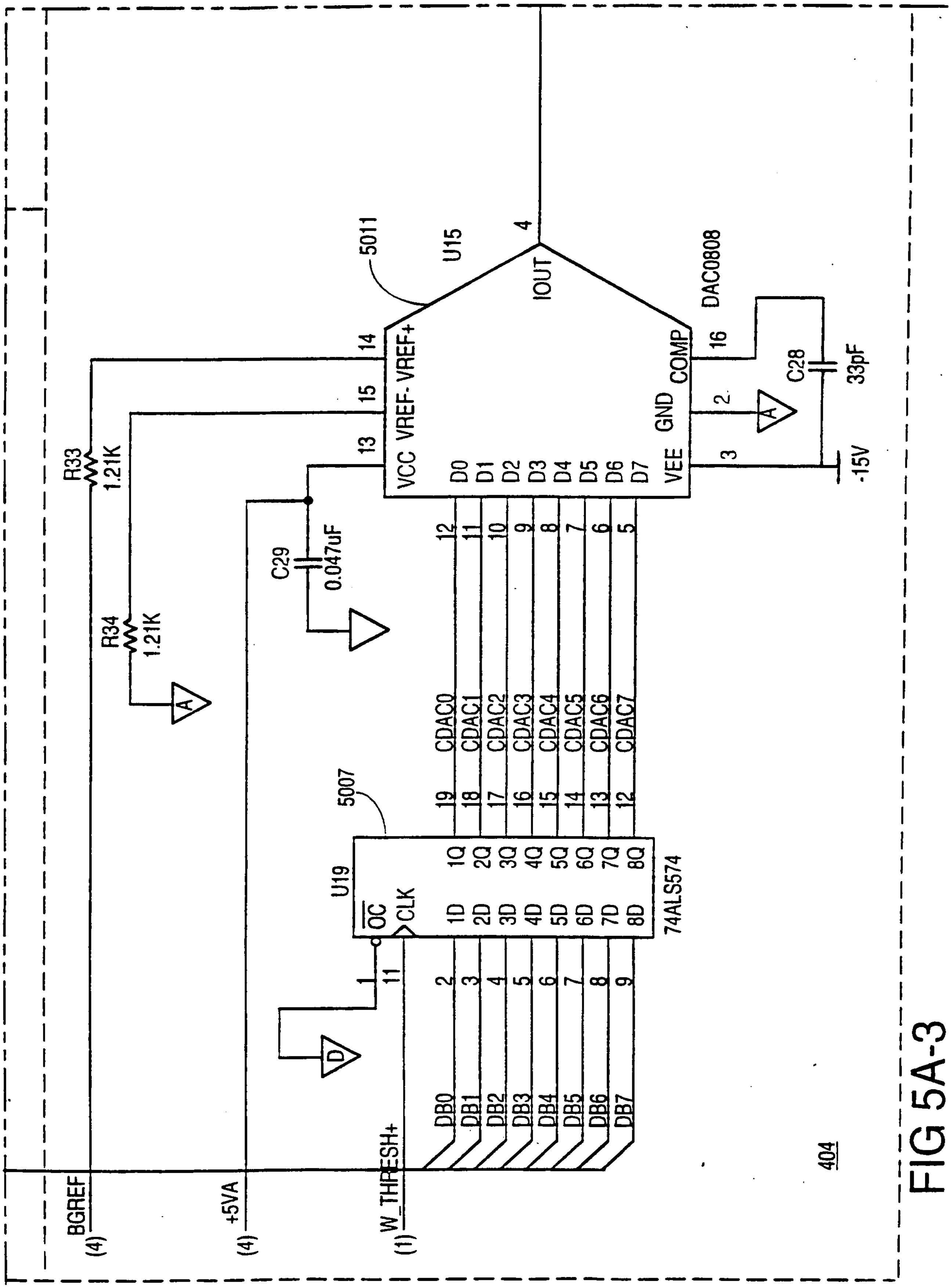

5011
U15
4
IOUT
DAC0808
VCC VREF- VREF+
14
15
13
16
COMP
2
GND
3
VEE
C28
33pF
-15V
A
A
D0 D1 D2 D3 D4 D5 D6 D7
12 11 10 9 8 7 6 5
R33
1.21K
R34
1.21K
C29
0.047uF
CDAC0 CDAC1 CDAC2 CDAC3 CDAC4 CDAC5 CDAC6 CDAC7
5007
19 18 17 16 15 14 13 12
U19
OC
CLK
1D 2D 3D 4D 5D 6D 7D 8D
1Q 2Q 3Q 4Q 5Q 6Q 7Q 8Q
74ALS574
1
11
2 3 4 5 6 7 8 9
D
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7
BGREF
(4)
+5VA
(4)
W_THRESH+
(1)
404

FIG 5A-3

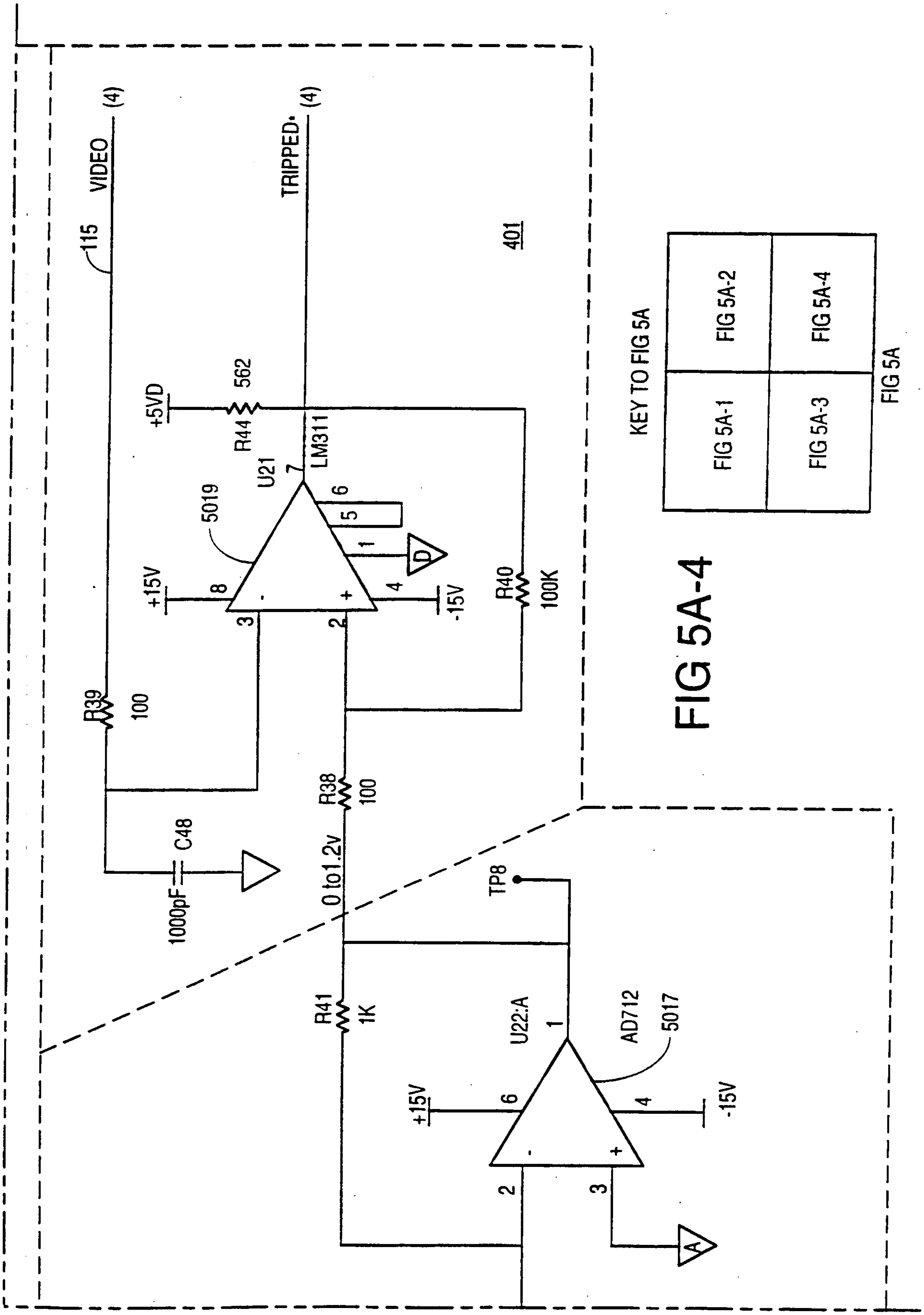
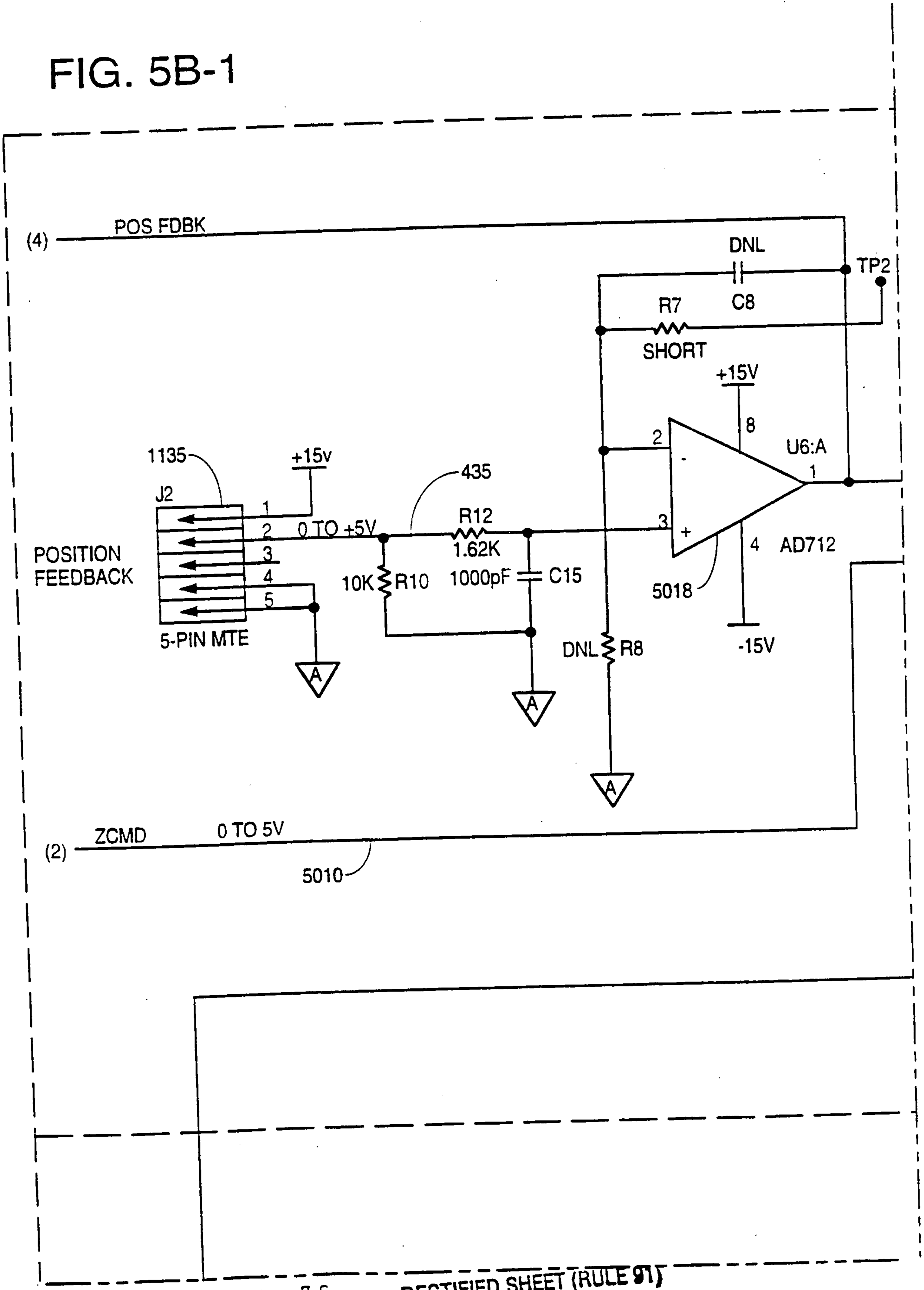

VIDEO
(4)
115
TRIPPED
(4)
401
+5VD
R44
562
U21
LM311
7
6
5
1
D
5019
+15V
8
-
+
4
-15V
3
2
R40
100K
R39
100
R38
100
C48
1000pF
0 to1.2v
TP8
R41
1K
U22:A
1
AD712
5017
+15V
6
4
-15V
2
3
A
KEY TO FIG 5A
FIG 5A-1
FIG 5A-2
FIG 5A-3
FIG 5A-4
FIG 5A

FIG 5A-4

FIG. 5B-1


(4)
POS FDBK
DNL
C8
TP2
R7
SHORT
+15V
8
2
-
U6:A
1
3
+
4
AD712
5018
-15V
DNL
R8
A
1135
+15v
J2
1
2
0 TO +5V
3
4
5
POSITION FEEDBACK
5-PIN MTE
435
R12
1.62K
10K
R10
1000pF
C15
A
A
(2)
ZCMD
0 TO 5V
5010

RECTIFIED SHEET (RULE 91)

FIG. 5B-2

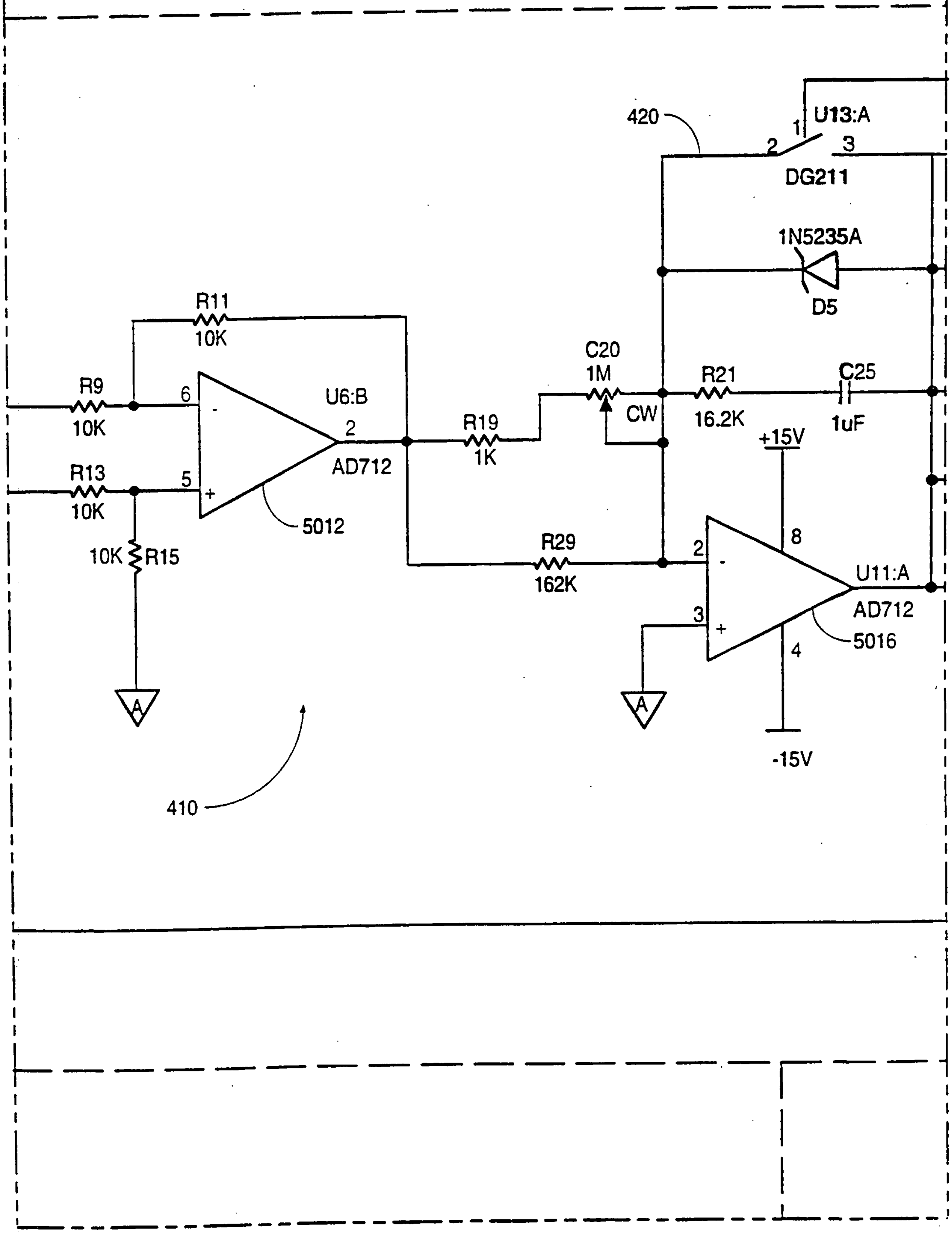
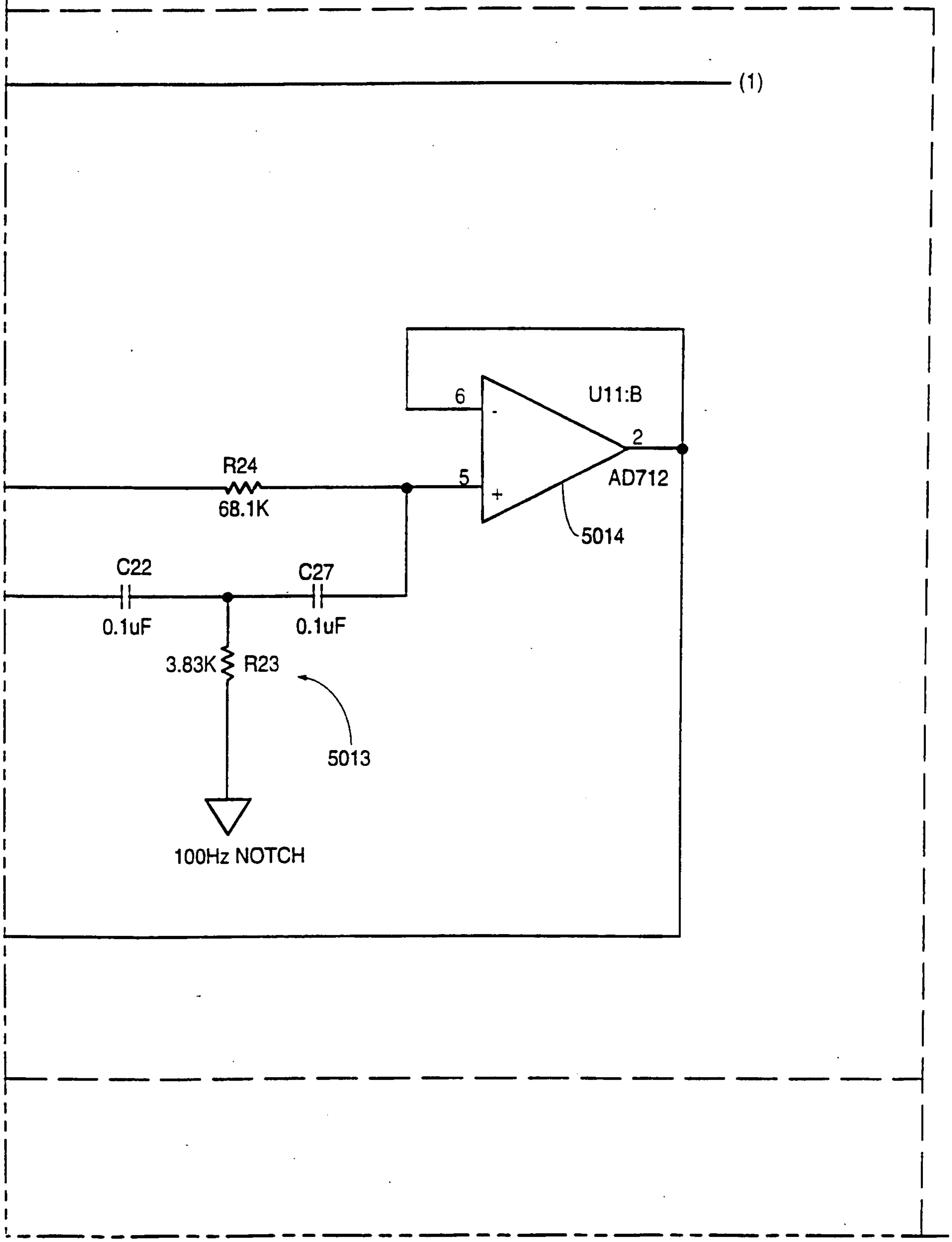

420
U13:A
1
2
3
DG211
1N5235A
D5
R11
10K
C20
1M
R21
16.2K
C25
1uF
R9
10K
6
U6:B
2
AD712
R19
1K
CW
+15V
R13
10K
5
5012
R29
162K
8
2
U11:A
AD712
3
5016
4
10K
R15
A
A
-15V
410

FIG. 5B-3


(1)
6
-
U11:B
2
R24
5
AD712
68.1K
+
5014
C22
C27
0.1uF
0.1uF
3.83K
R23
5013
100Hz NOTCH

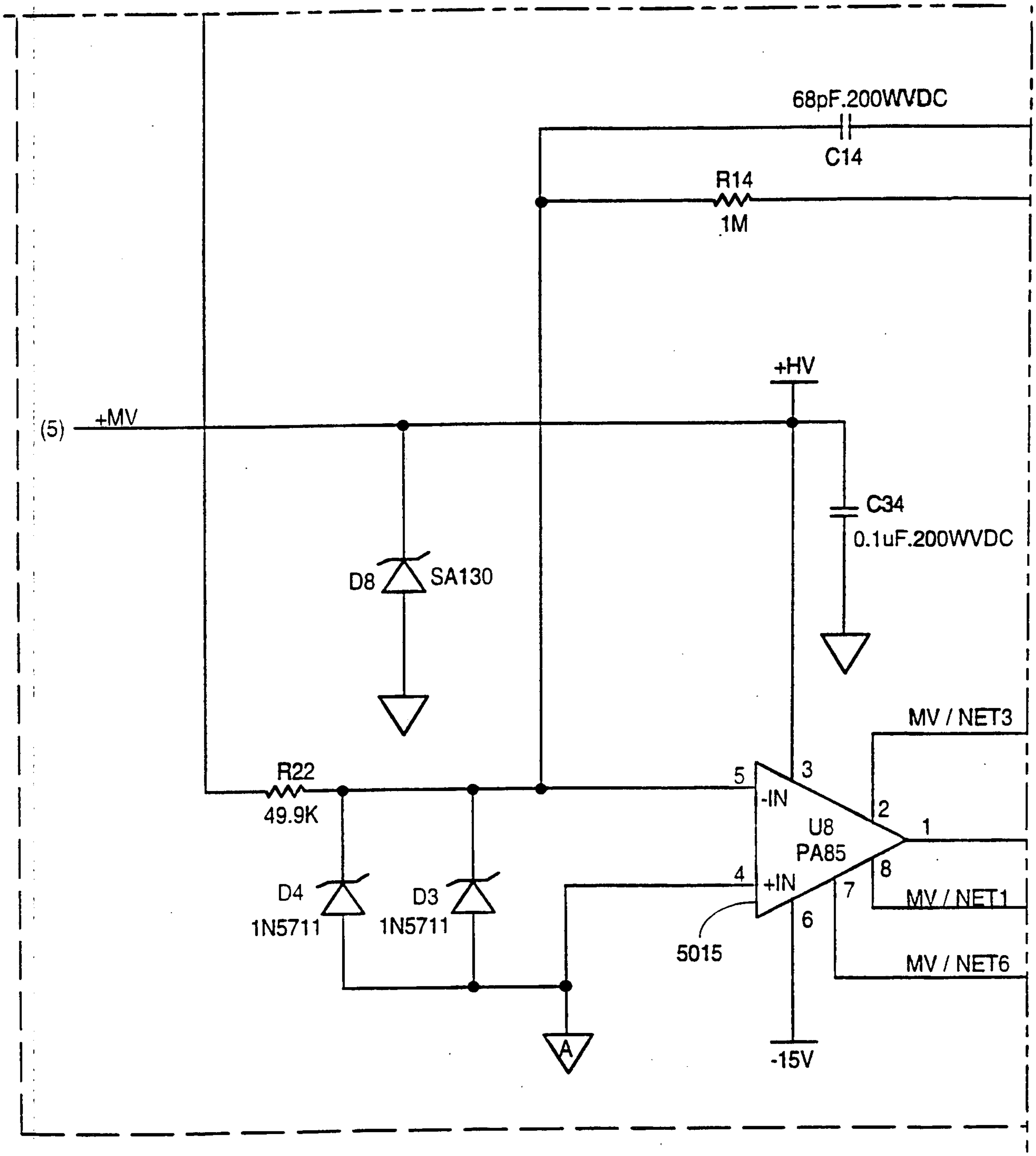

68pF.200WVDC
C14
R14
1M
+HV
(5)
+MV
C34
0.1uF.200WVDC
D8
SA130
MV / NET3
R22
49.9K
5
-IN
3
U8
PA85
2
1
8
4
+IN
7
MV / NET1
D4
1N5711
D3
1N5711
5015
6
MV / NET6
A
-15V

FIG. 5B-4

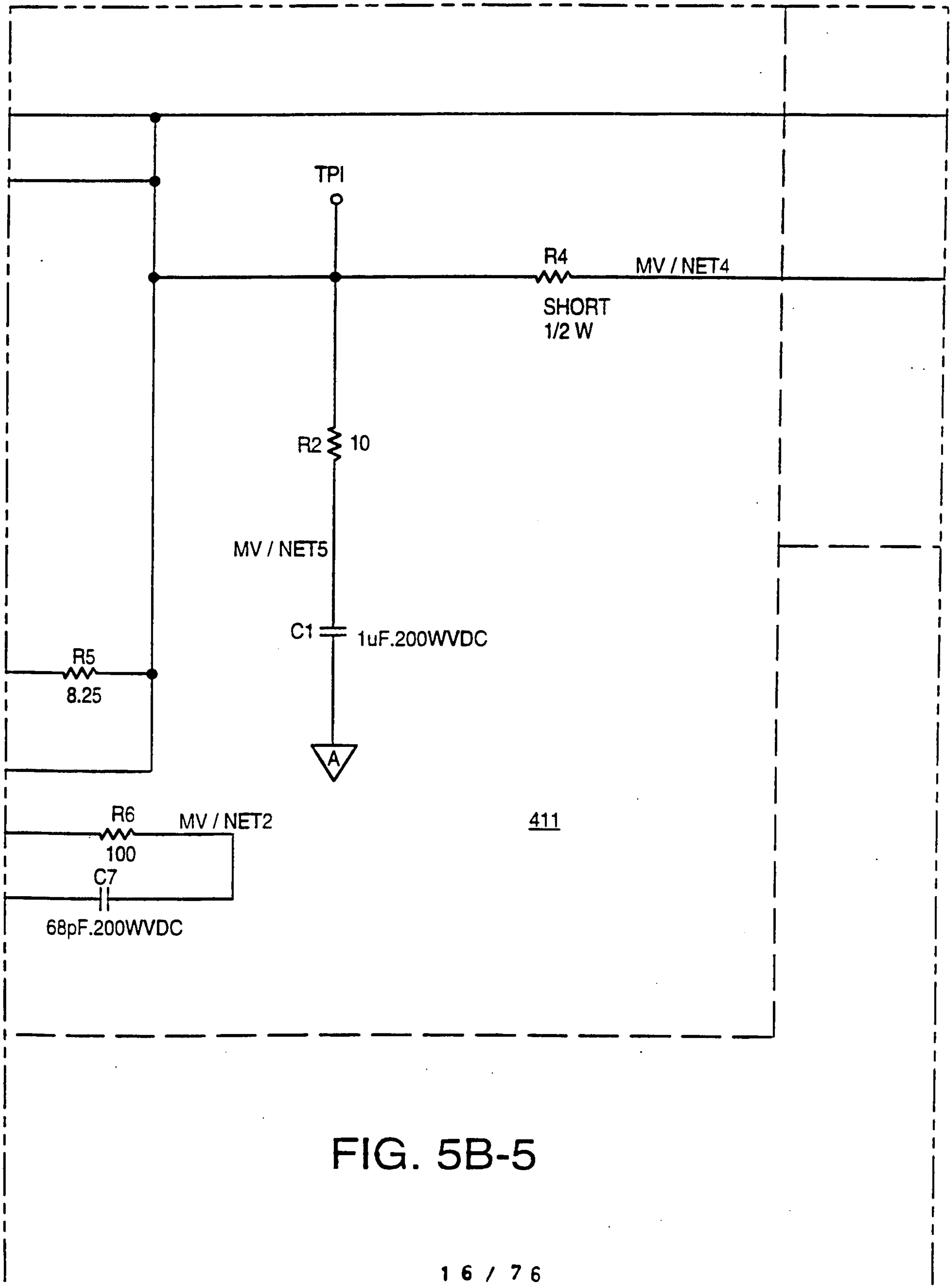

TPI
R4
MV / NET4
SHORT
1/2 W
R2
10
MV / NET5
C1
1uF.200WVDC
R5
8.25
A
R6
MV / NET2
100
C7
68pF.200WVDC
411

FIG. 5B-5

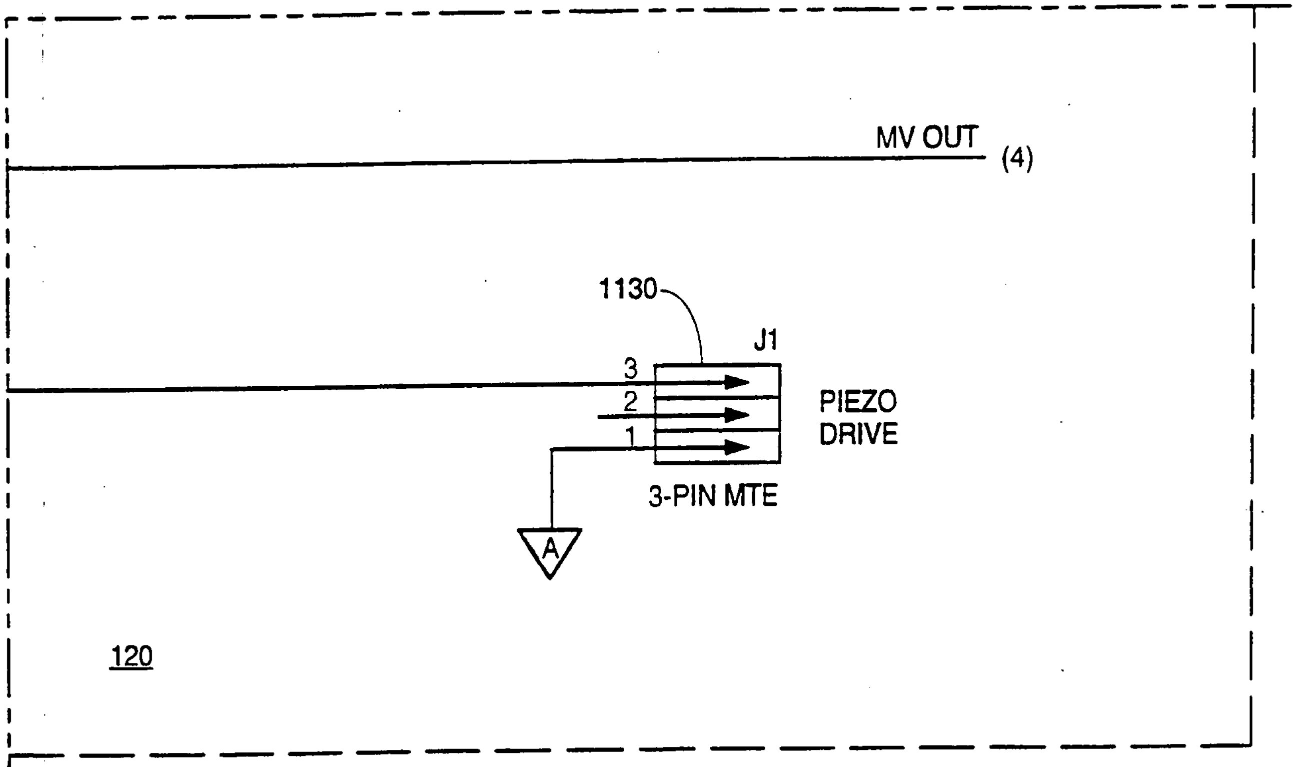

MV OUT
(4)
1130
J1
3
2
1
PIEZO
DRIVE
3-PIN MTE
A
120

FIG. 5B-6

KEY TO FIG. 5B

FIG. 5B-1	FIG. 5B-2	FIG. 5B-3
FIG. 5B-4	FIG. 5B-5	FIG. 5B-6

FIG. 5B

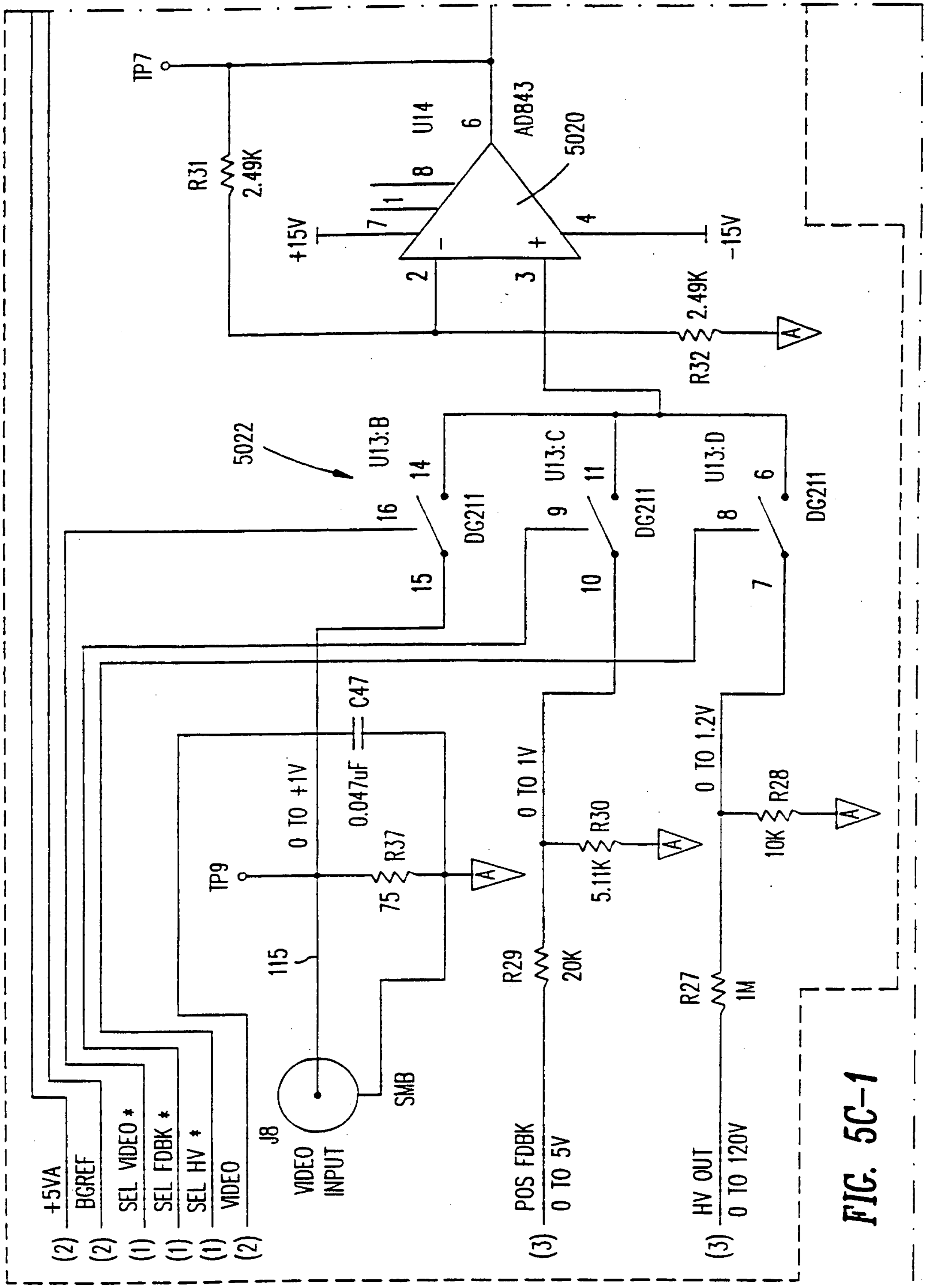

TP7
R31
2.49K
U14
6
AD843
5020
8
1
7
+15V
4
-15V
2
3
R32
2.49K
A
5022
U13:B
14
16
15
DG211
U13:C
11
9
10
DG211
U13:D
6
8
7
DG211
C47
0.047uF
0 TO +1V
0 TO 1V
0 TO 1.2V
TP9
R37
75
A
R30
5.11K
A
R28
10K
A
115
R29
20K
R27
1M
J8
VIDEO INPUT
SMB
(2) +5VA
(2) BGREF
(1) SEL VIDEO *
(1) SEL FDBK *
(1) SEL HV *
(2) VIDEO
(3) POS FDBK 0 TO 5V
(3) HV OUT 0 TO 120V

FIG. 5C-1

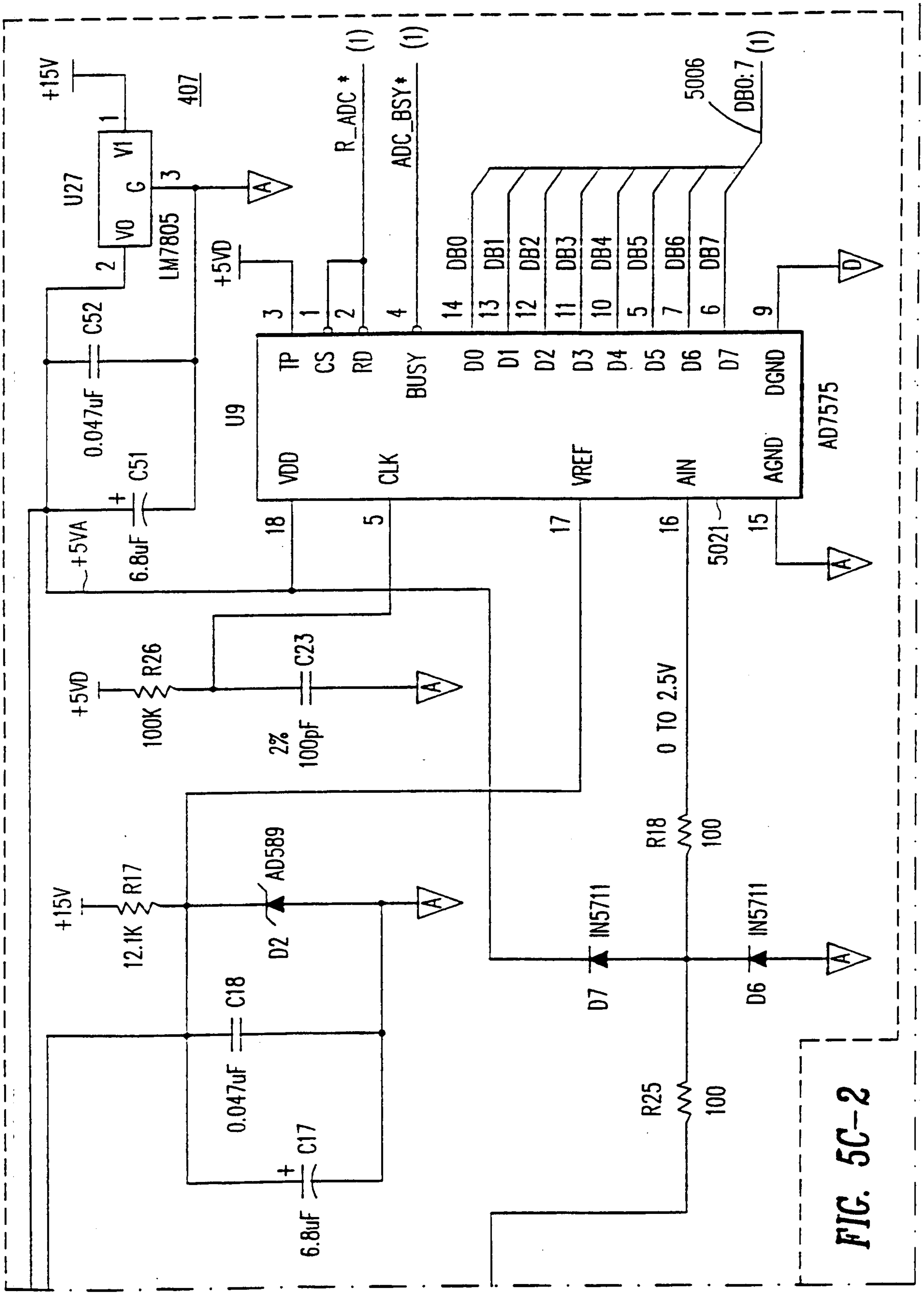

+15V
407
U27
V1
G
VO
1
3
2
LM7805
A
R_ADC *
ADC_BSY *
(1)
(1)
(1)
5006
DB0:7
+5VD
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
3
1
2
4
14
13
12
11
10
5
7
6
9
D
C52
0.047uF
C51
6.8uF
+5VA
U9
TP
CS
RD
BUSY
D0
D1
D2
D3
D4
D5
D6
D7
DGND
AD7575
VDD
CLK
VREF
AIN
AGND
18
5
17
16
5021
15
A
+5VD
R26
100K
C23
2%
100pF
A
0 TO 2.5V
R18
100
+15V
R17
12.1K
AD589
D2
A
IN5711
D7
IN5711
D6
A
C18
0.047uF
C17
6.8uF
R25
100

FIG. 5C-2

RECTIFIED SHEET (RULE 91)

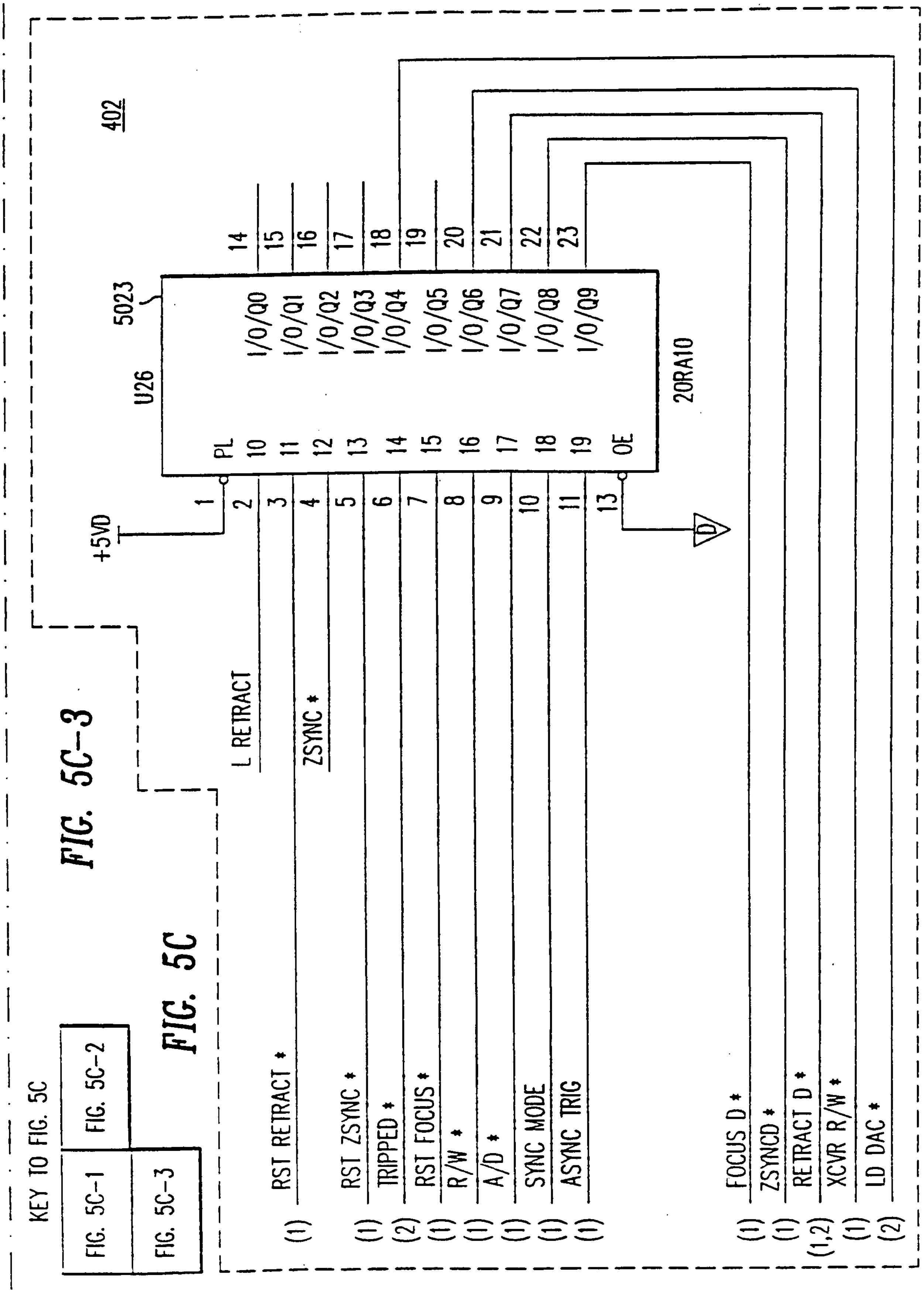

402
KEY TO FIG. 5C
FIG. 5C-1
FIG. 5C-2
FIG. 5C-3
5023
U26
20RA10
+5VD
PL
OE
I/O/Q0
I/O/Q1
I/O/Q2
I/O/Q3
I/O/Q4
I/O/Q5
I/O/Q6
I/O/Q7
I/O/Q8
I/O/Q9
1
2
3
4
5
6
7
8
9
10
11
13
14
15
16
17
18
19
20
21
22
23
D
L RETRACT
ZSYNC *
(1) RST RETRACT *
(1) RST ZSYNC *
(2) TRIPPED *
(1) RST FOCUS *
(1) R/W *
(1) A/D *
(1) SYNC MODE
(1) ASYNC TRIG
(1) FOCUS D *
(1) ZSYNCD *
(1,2) RETRACT D *
(1) XCVR R/W *
(2) LD DAC *

FIG. 5C-3

FIG. 5C

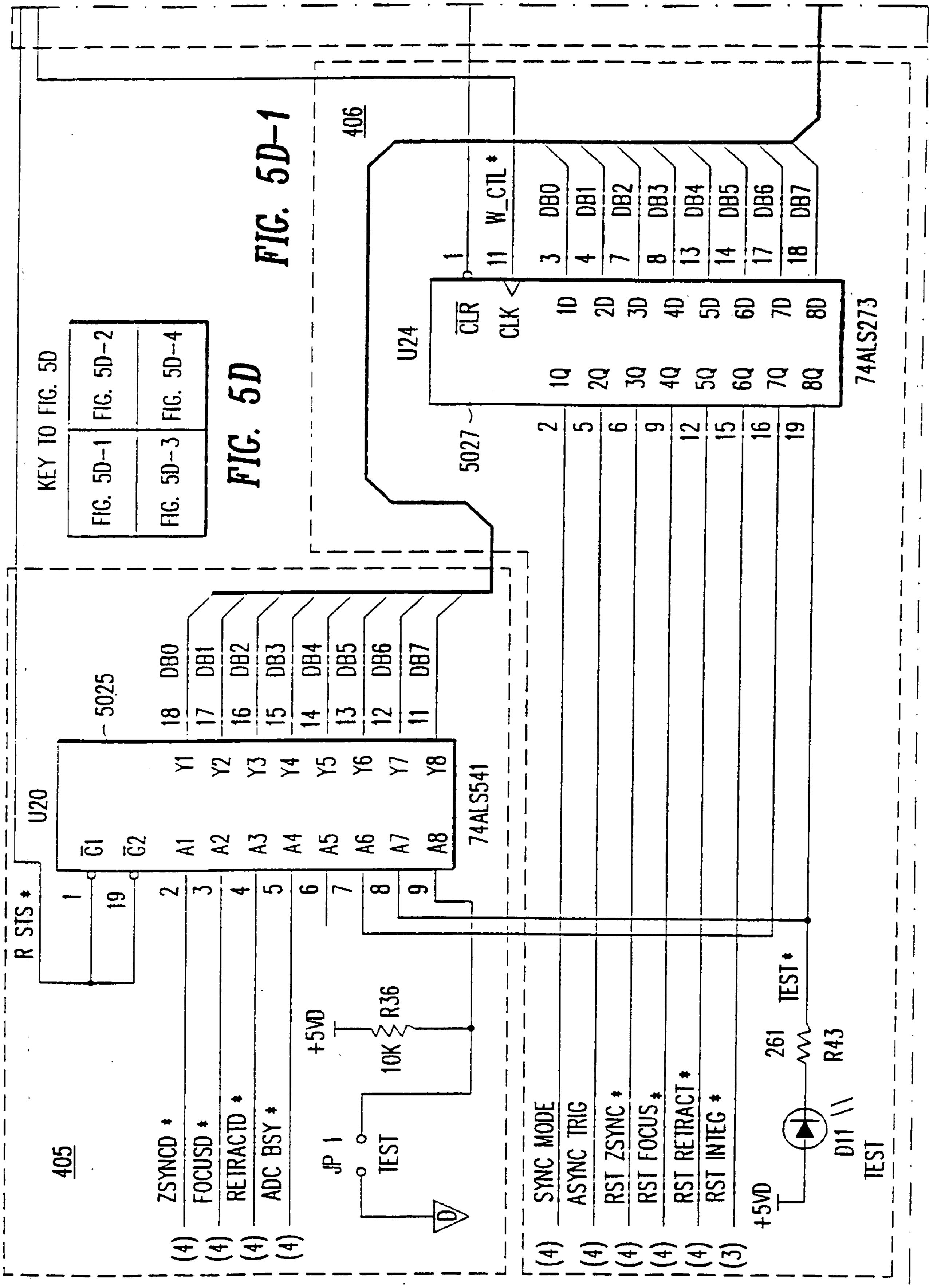

KEY TO FIG. 5D
FIG. 5D-1
FIG. 5D-2
FIG. 5D-3
FIG. 5D-4
406
U24
CLR
CLK
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
74ALS273
5027
1
11 W_CTL *
3 DB0
4 DB1
7 DB2
8 DB3
13 DB4
14 DB5
17 DB6
18 DB7
2
5
6
9
12
15
16
19
405
U20
5025
G1
G2
A1
A2
A3
A4
A5
A6
A7
A8
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
74ALS541
18 DB0
17 DB1
16 DB2
15 DB3
14 DB4
13 DB5
12 DB6
11 DB7
R STS *
1
19
2
3
4
5
6
7
8
9
(4) ZSYNCD *
(4) FOCUSD *
(4) RETRACTD *
(4) ADC BSY *
JP 1
TEST
D
+5VD
10K
R36
(4) SYNC MODE
(4) ASYNC TRIG
(4) RST ZSYNC *
(4) RST FOCUS *
(4) RST RETRACT *
(4) RST INTEG *
(3) +5VD
TEST *
261
R43
D11
TEST

FIG. 5D

FIG. 5D-1

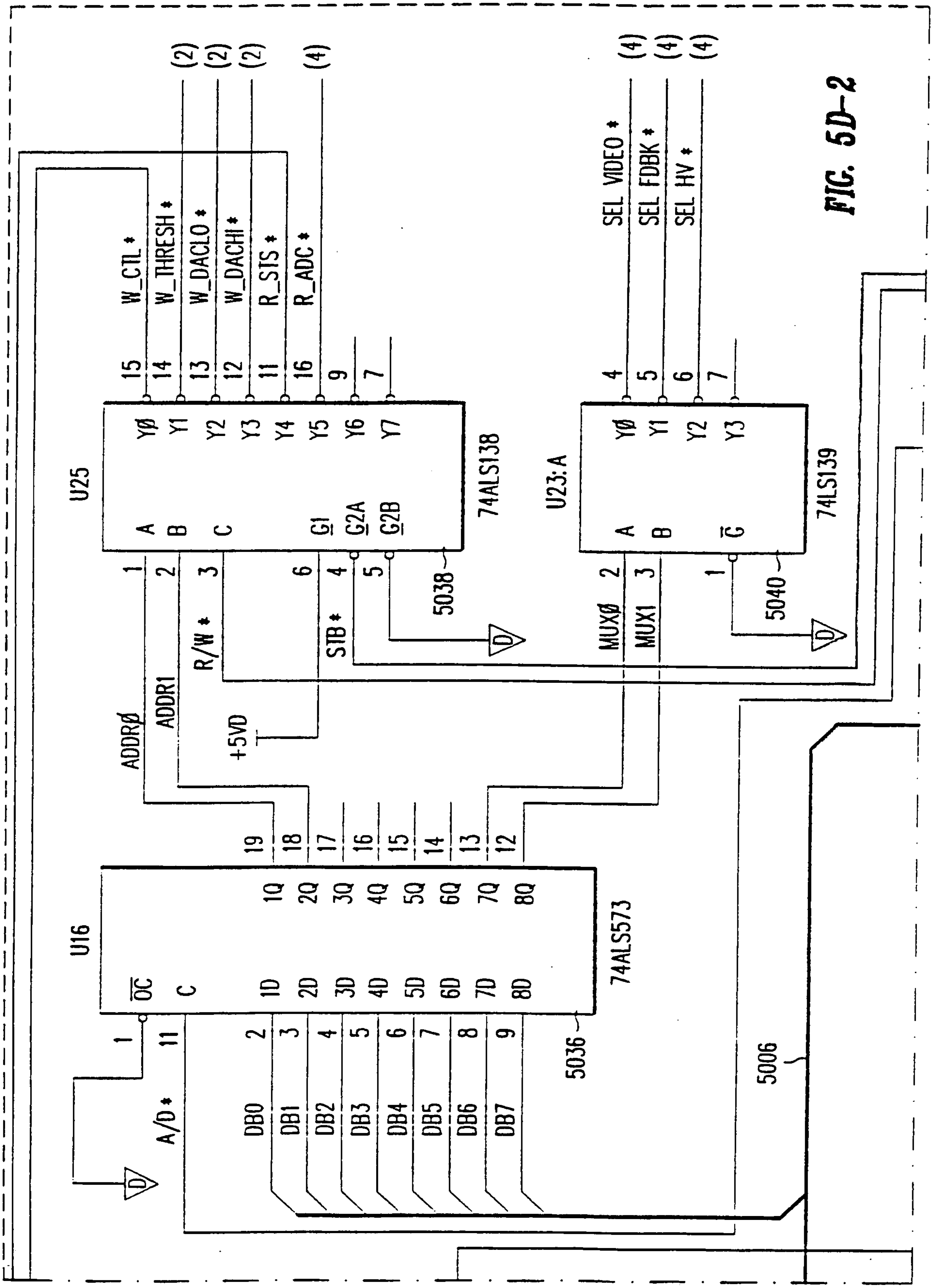

W_CTL*
W_THRESH*
W_DACLO*
W_DACHI*
R_STS*
R_ADC*
(2)
(2)
(2)
(4)
SEL VIDEO*
SEL FDBK*
SEL HV*
(4)
(4)
(4)
U25
74ALS138
A
B
C
G1
G2A
G2B
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
15
14
13
12
11
16
9
7
1
2
3
6
4
5
5038
U23:A
74LS139
A
B
G
Y0
Y1
Y2
Y3
4
5
6
7
2
3
1
5040
MUX0
MUX1
R/W*
STB*
ADDR0
ADDR1
+5VD
D
D
D
U16
74ALS573
OC
C
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
19
18
17
16
15
14
13
12
1
11
2
3
4
5
6
7
8
9
5036
A/D*
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
5006

FIG. 5D-2

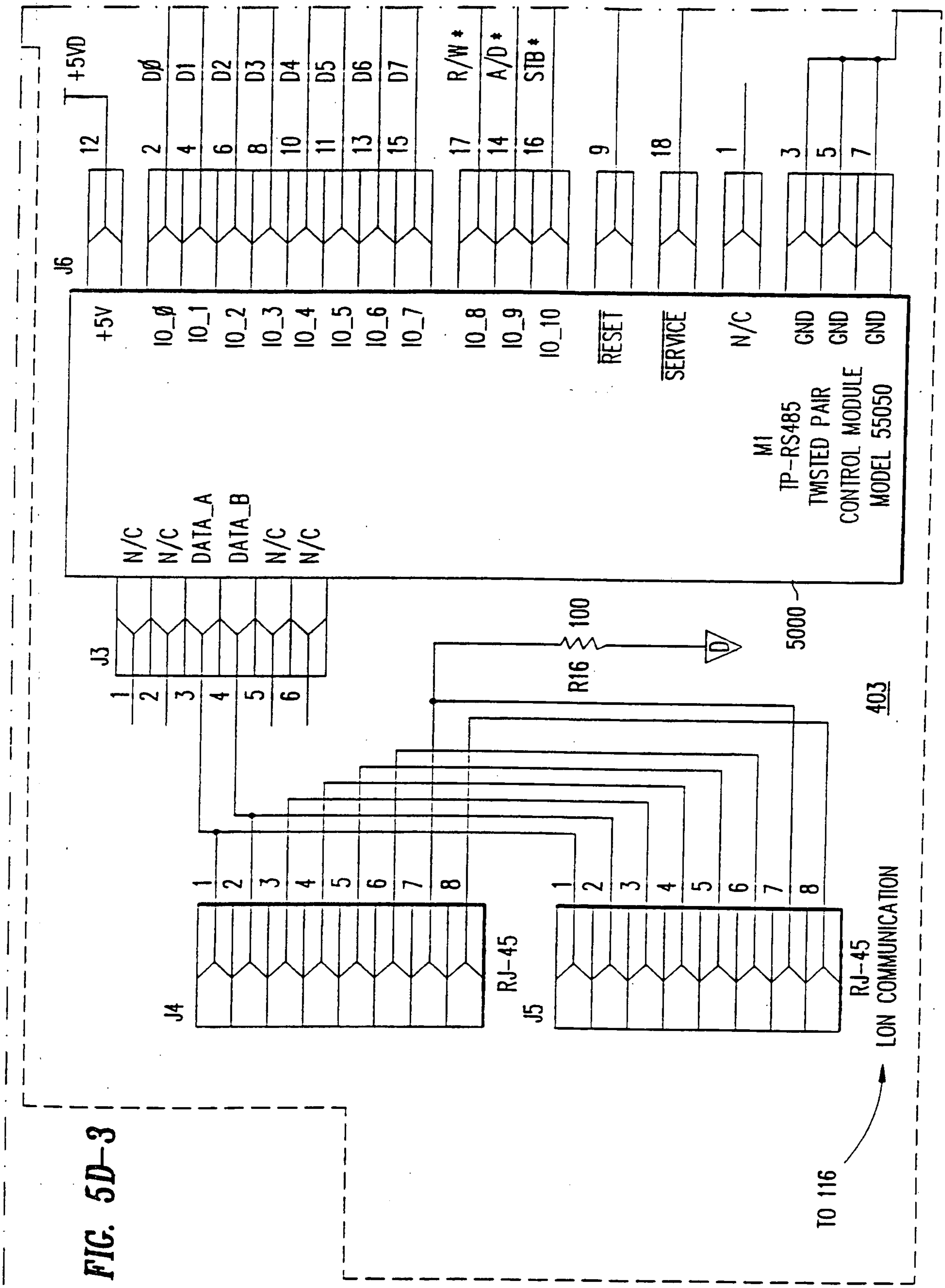

+5VD
12
D0
D1
D2
D3
D4
D5
D6
D7
2
4
6
8
10
11
13
15
R/W*
A/D*
STB*
17
14
16
9
18
1
3
5
7
J6
+5V
IO_0
IO_1
IO_2
IO_3
IO_4
IO_5
IO_6
IO_7
IO_8
IO_9
IO_10
RESET
SERVICE
N/C
GND
GND
GND
M1
TP-RS485
TWISTED PAIR
CONTROL MODULE
MODEL 55050
N/C
N/C
DATA_A
DATA_B
N/C
N/C
5000
J3
1
2
3
4
5
6
R16
100
D
403
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
J4
RJ-45
J5
RJ-45
LON COMMUNICATION
TO 116

FIG. 5D-3

RECTIFIED SHEET (RULE 91)

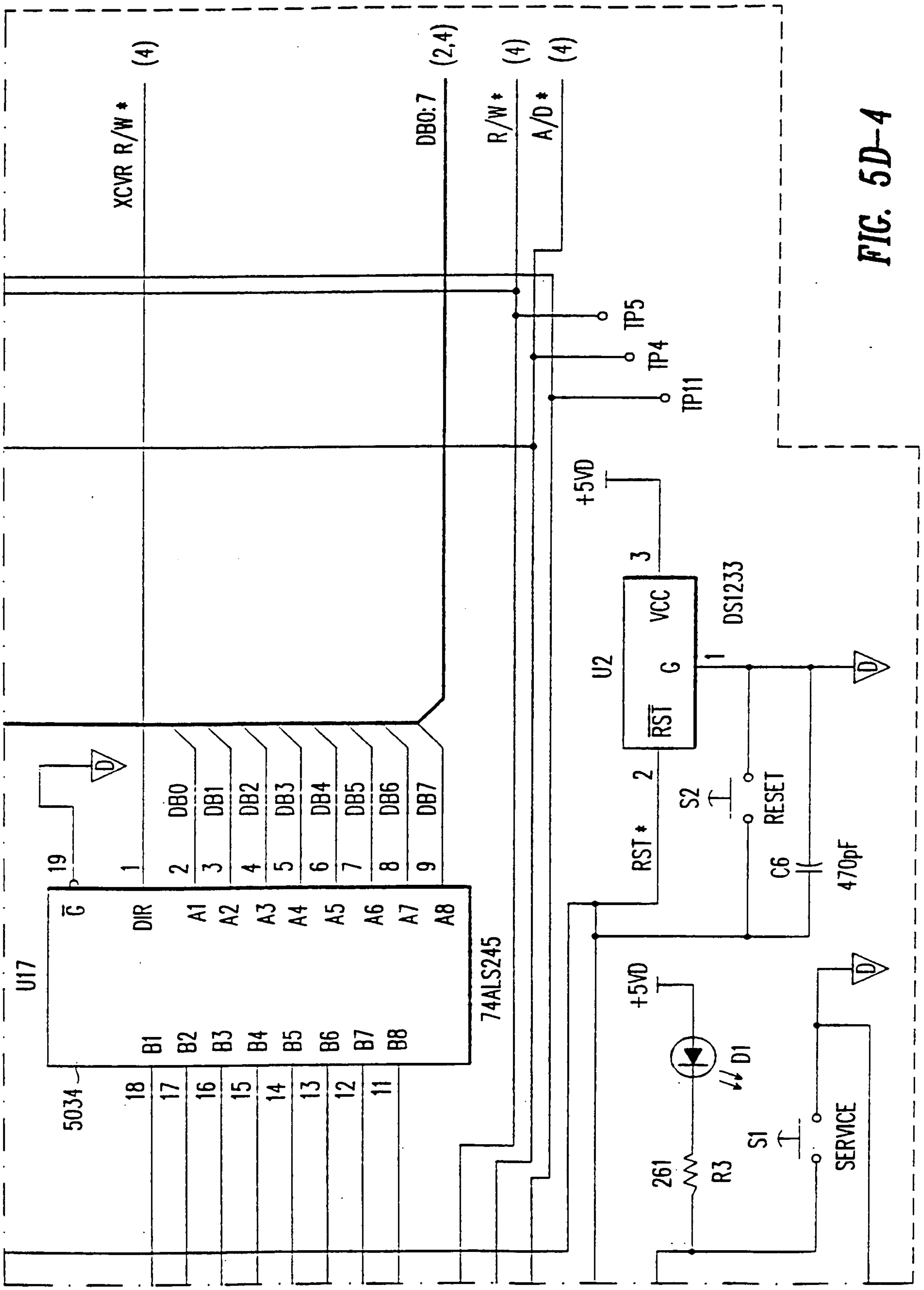

XCVR R/W *
(4)
DB0:7
(2,4)
R/W *
(4)
A/D *
(4)
TP5
TP4
TP11
+5VD
3
DS1233
VCC
U2
G
1
D
RST
2
RST *
S2
RESET
C6
470pF
U17
G
19
D
DIR
1
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
2
3
4
5
6
7
8
9
A1
A2
A3
A4
A5
A6
A7
A8
74ALS245
B1
B2
B3
B4
B5
B6
B7
B8
5034
18
17
16
15
14
13
12
11
+5VD
D1
261
R3
S1
SERVICE
D

FIG. 5D-4

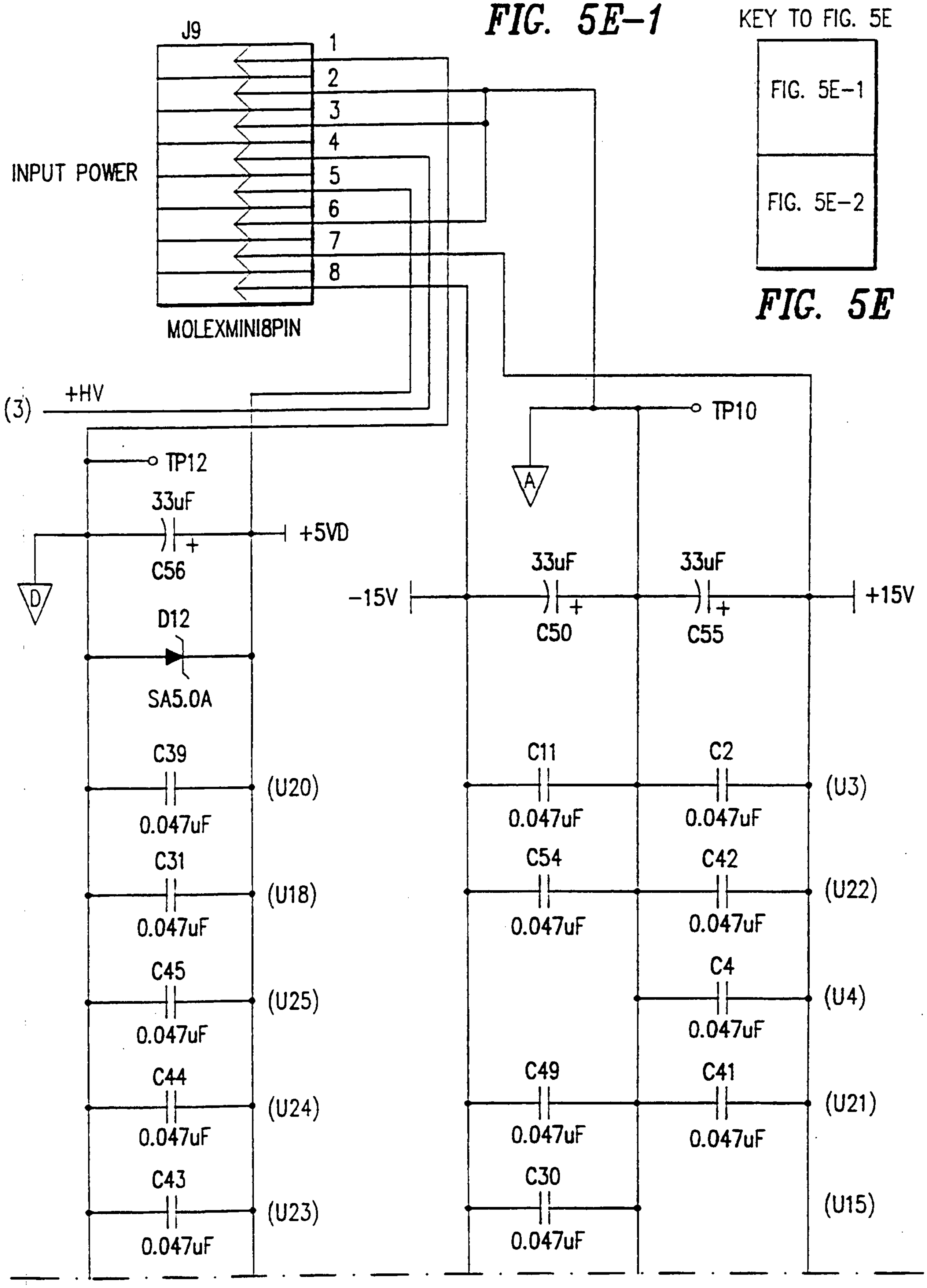

J9
1
2
3
4
5
6
7
8
INPUT POWER
MOLEXMINI8PIN
(3)
+HV
TP12
33uF
+
C56
+5VD
D
D12
SA5.0A
C39
0.047uF
(U20)
C31
0.047uF
(U18)
C45
0.047uF
(U25)
C44
0.047uF
(U24)
C43
0.047uF
(U23)
TP10
A
-15V
33uF
+
C50
33uF
+
C55
+15V
C11
0.047uF
C2
0.047uF
(U3)
C54
0.047uF
C42
0.047uF
(U22)
C4
0.047uF
(U4)
C49
0.047uF
C41
0.047uF
(U21)
C30
0.047uF
(U15)

FIG. 5E-1

KEY TO FIG. 5E

FIG. 5E-1
FIG. 5E-2

FIG. 5E

RECTIFIED SHEET (RULE 91)

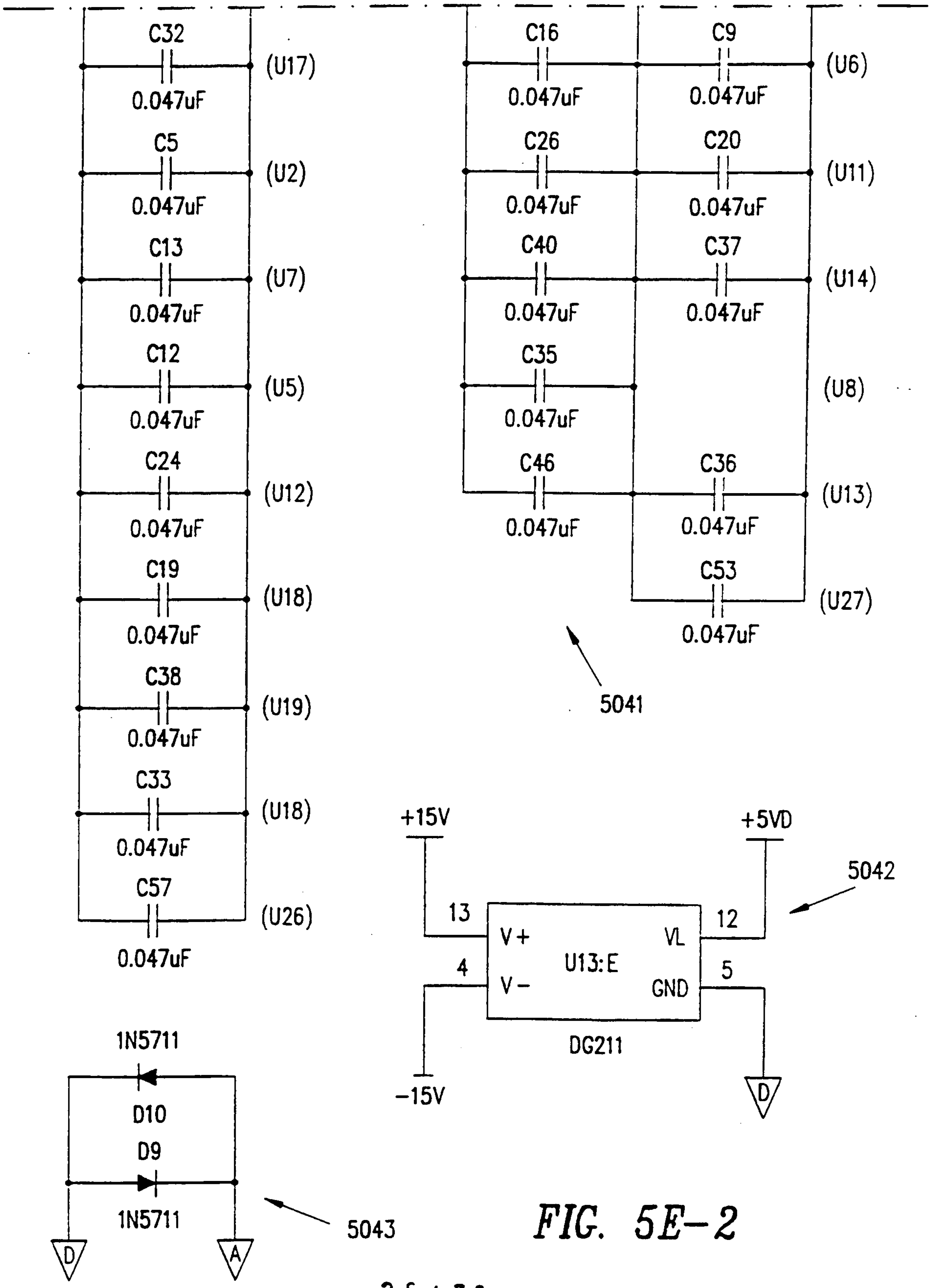

C32
0.047uF
(U17)
C5
0.047uF
(U2)
C13
0.047uF
(U7)
C12
0.047uF
(U5)
C24
0.047uF
(U12)
C19
0.047uF
(U18)
C38
0.047uF
(U19)
C33
0.047uF
(U18)
C57
0.047uF
(U26)
C16
0.047uF
C9
0.047uF
(U6)
C26
0.047uF
C20
0.047uF
(U11)
C40
0.047uF
C37
0.047uF
(U14)
C35
0.047uF
(U8)
C46
0.047uF
C36
0.047uF
(U13)
C53
0.047uF
(U27)
5041
+15V
+5VD
5042
13
V+
U13:E
VL
12
4
V−
GND
5
DG211
−15V
D
1N5711
D10
D9
1N5711
D
A
5043

FIG. 5E-2

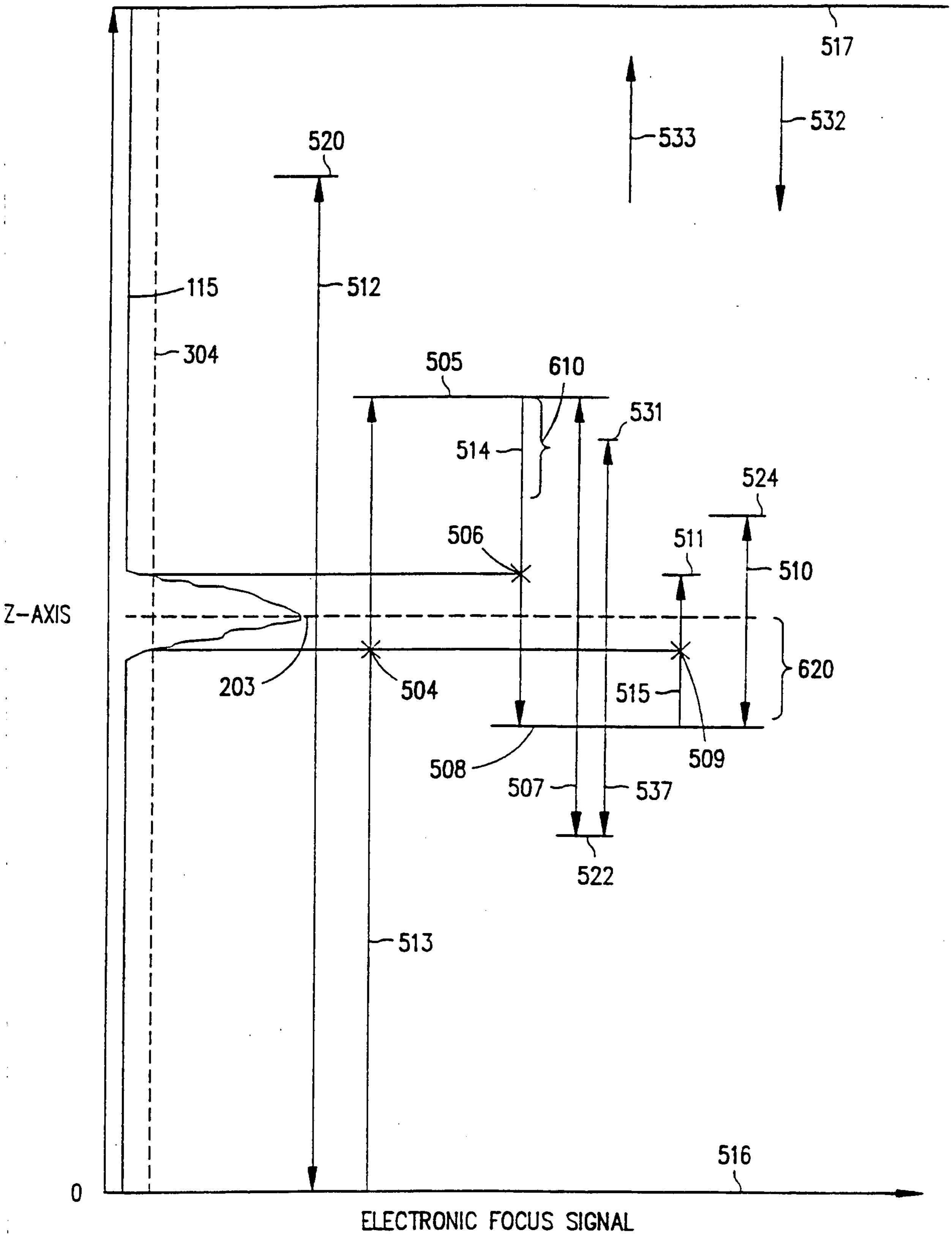
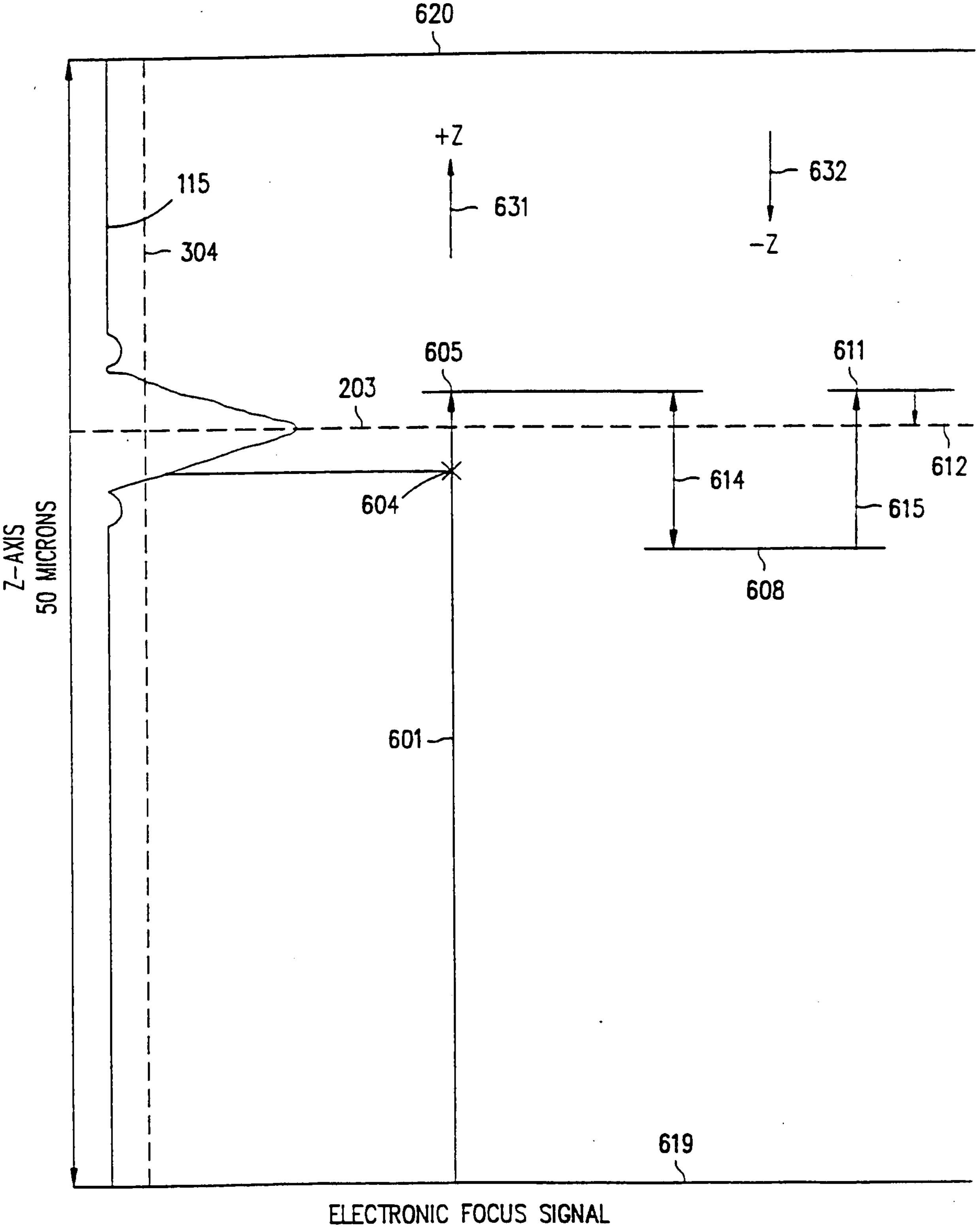

Z-AXIS
ELECTRONIC FOCUS SIGNAL

FIG. 6



620
+Z
631
632
-Z
115
304
605
611
203
612
604
614
615
608
Z-AXIS
50 MICRONS
601
619
ELECTRONIC FOCUS SIGNAL

FIG. 7

RECTIFIED SHEET (RULE 91)

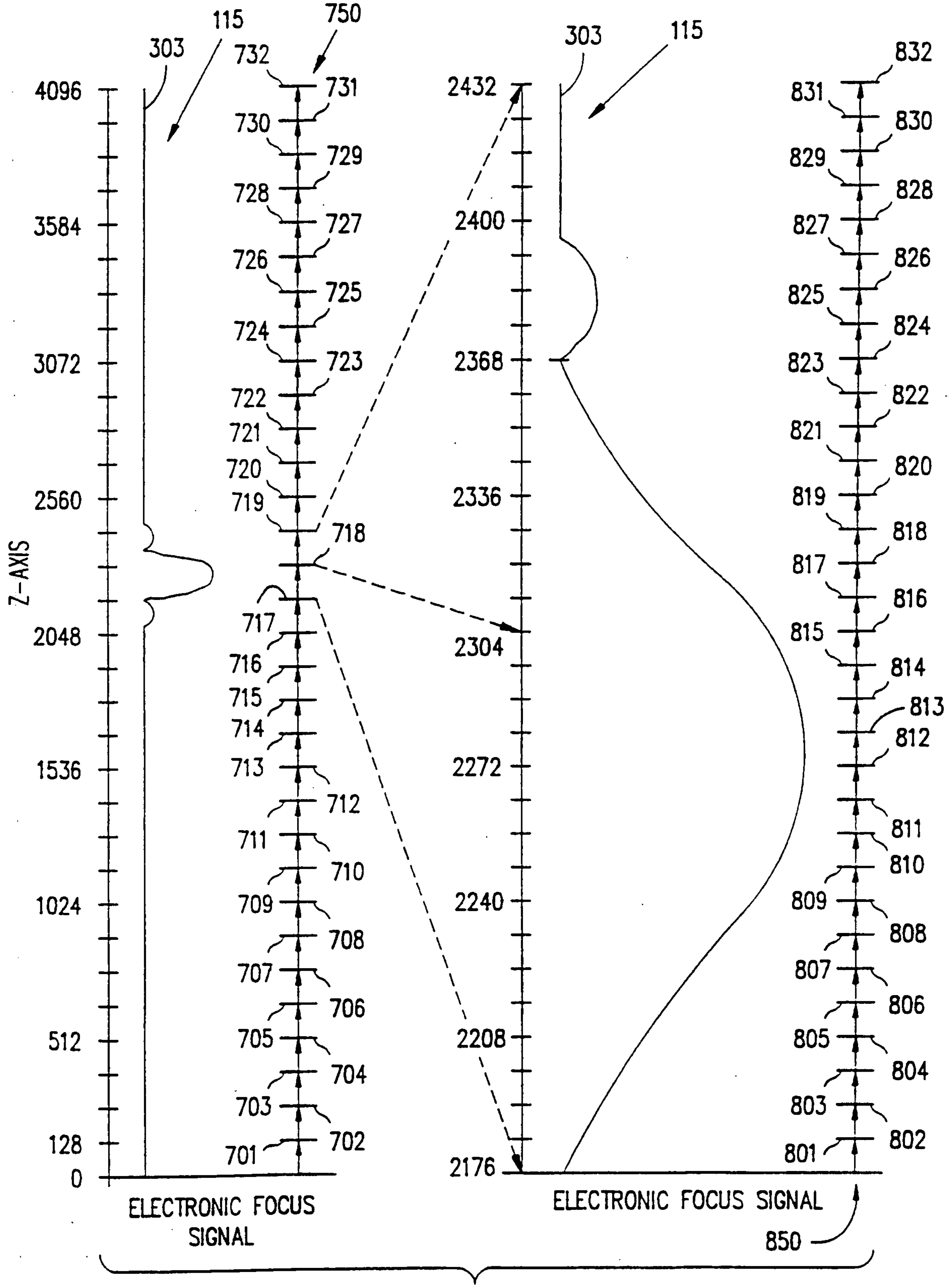

2048
1536
Z-AXIS
ELECTRONIC FOCUS SIGNAL
ELECTRONIC FOCUS SIGNAL

FIG. 8A

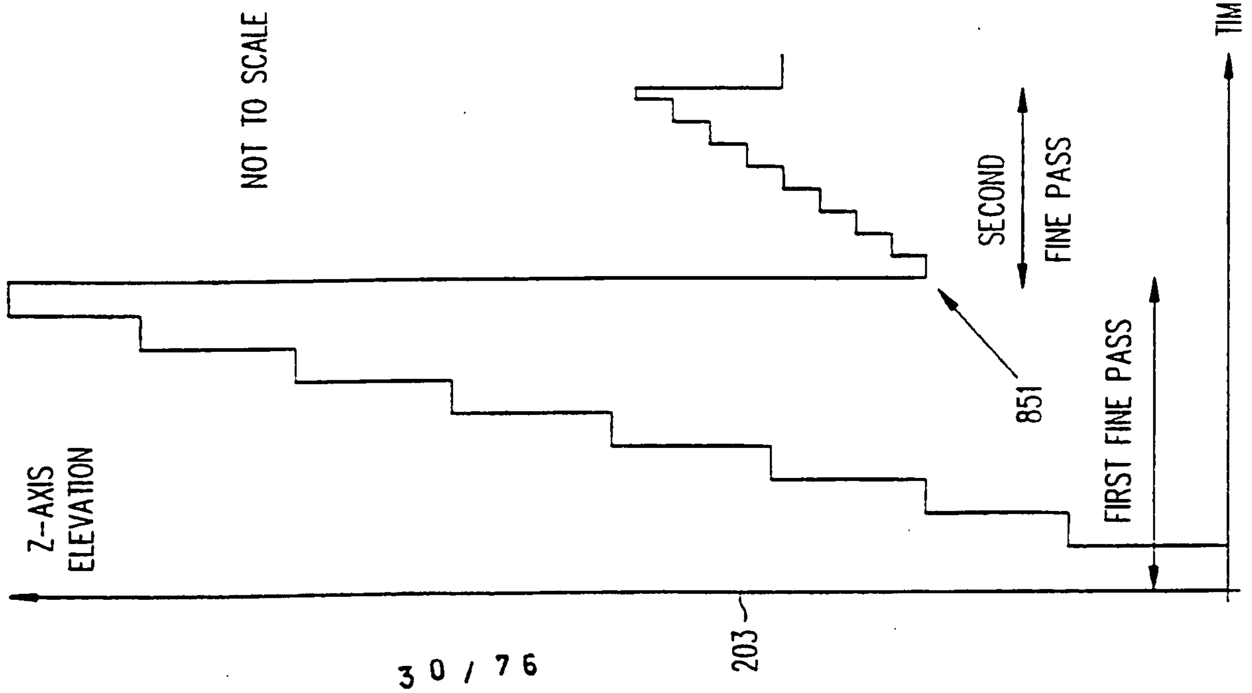

NOT TO SCALE
Z-AXIS ELEVATION
TIME
SECOND FINE PASS
FIRST FINE PASS
851
203

FIG. 8B

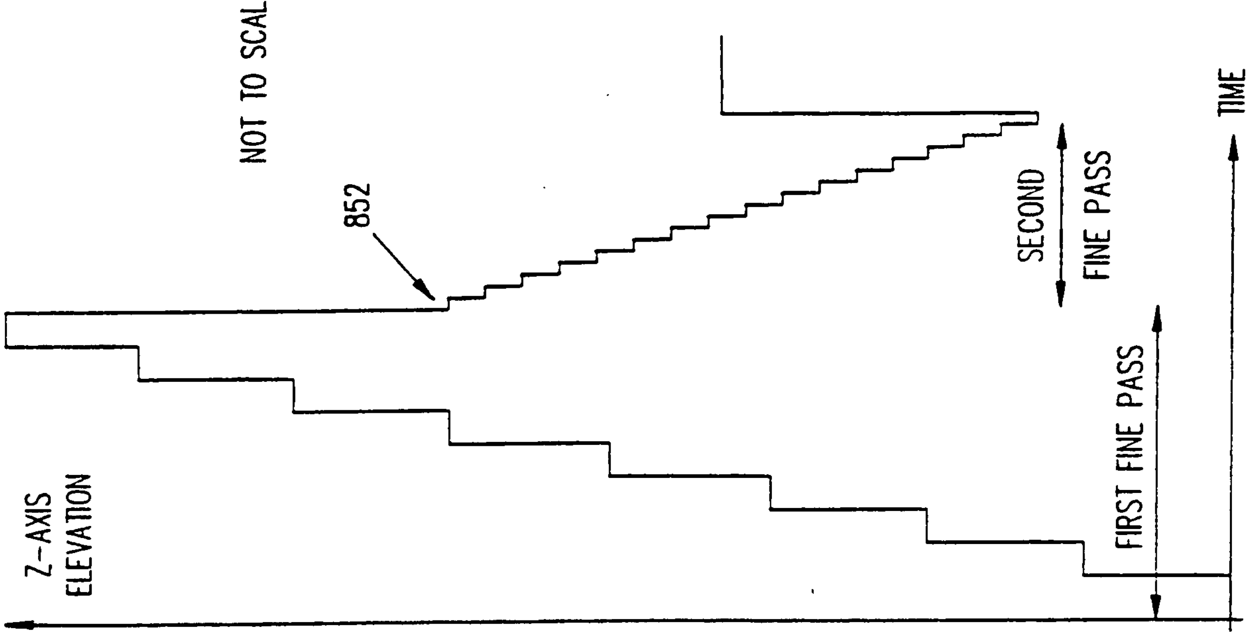

NOT TO SCALE
852
Z-AXIS ELEVATION
TIME
SECOND FINE PASS
FIRST FINE PASS

FIG. 8C

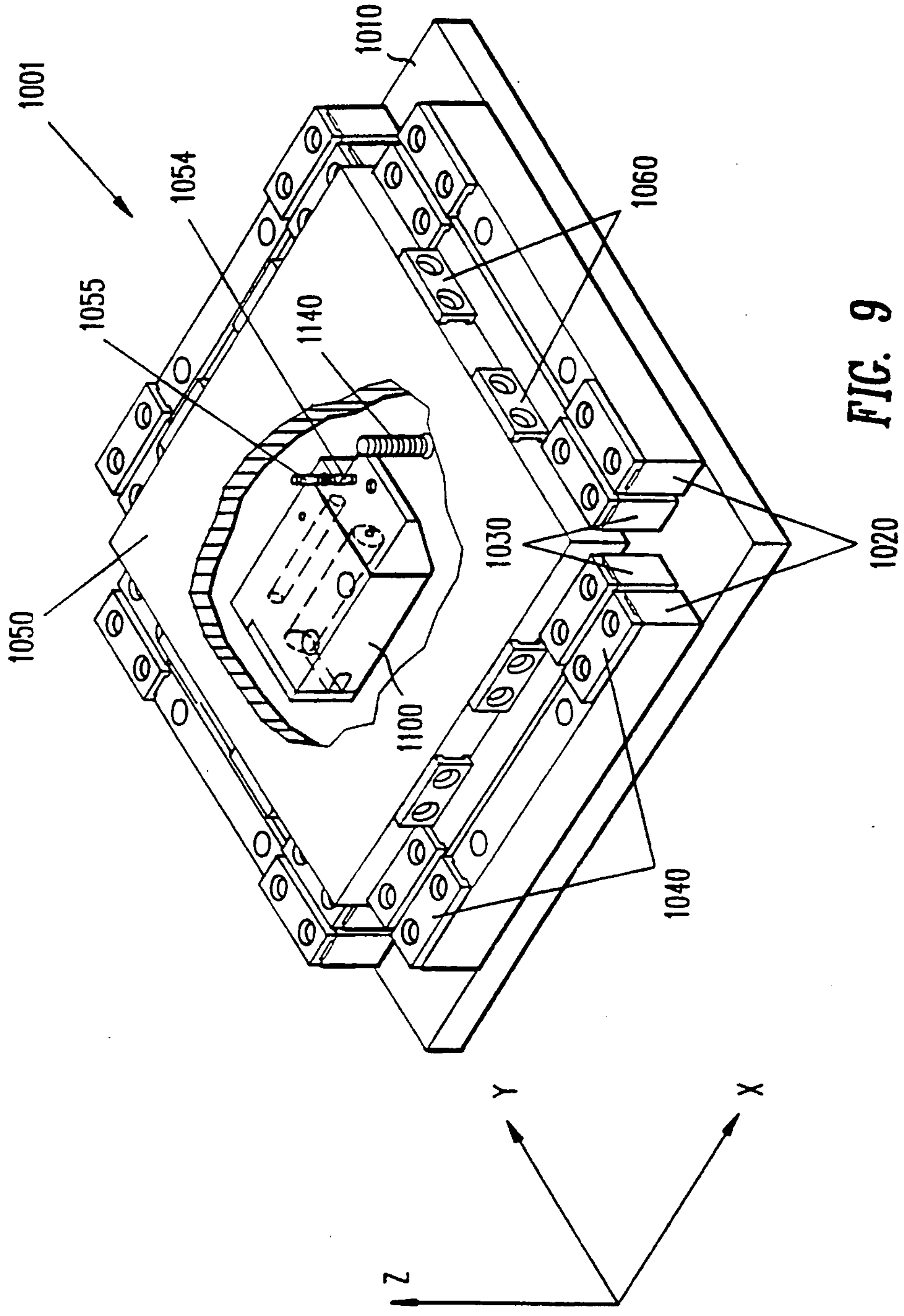

1001
1010
1054
1055
1140
1060
1050
1100
1030
1020
1040
Y
X
Z

FIG. 9

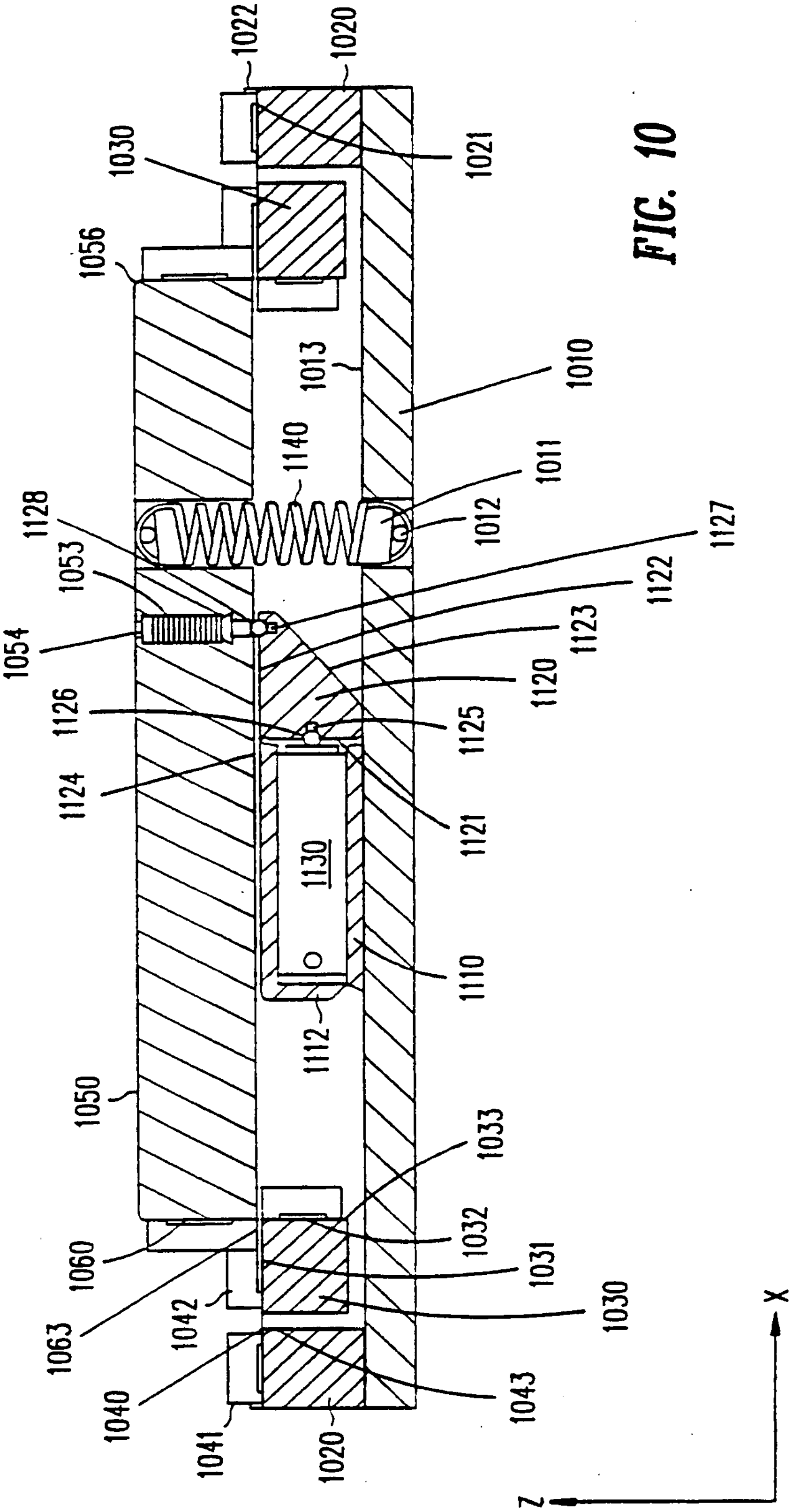

X
Z

FIG. 10

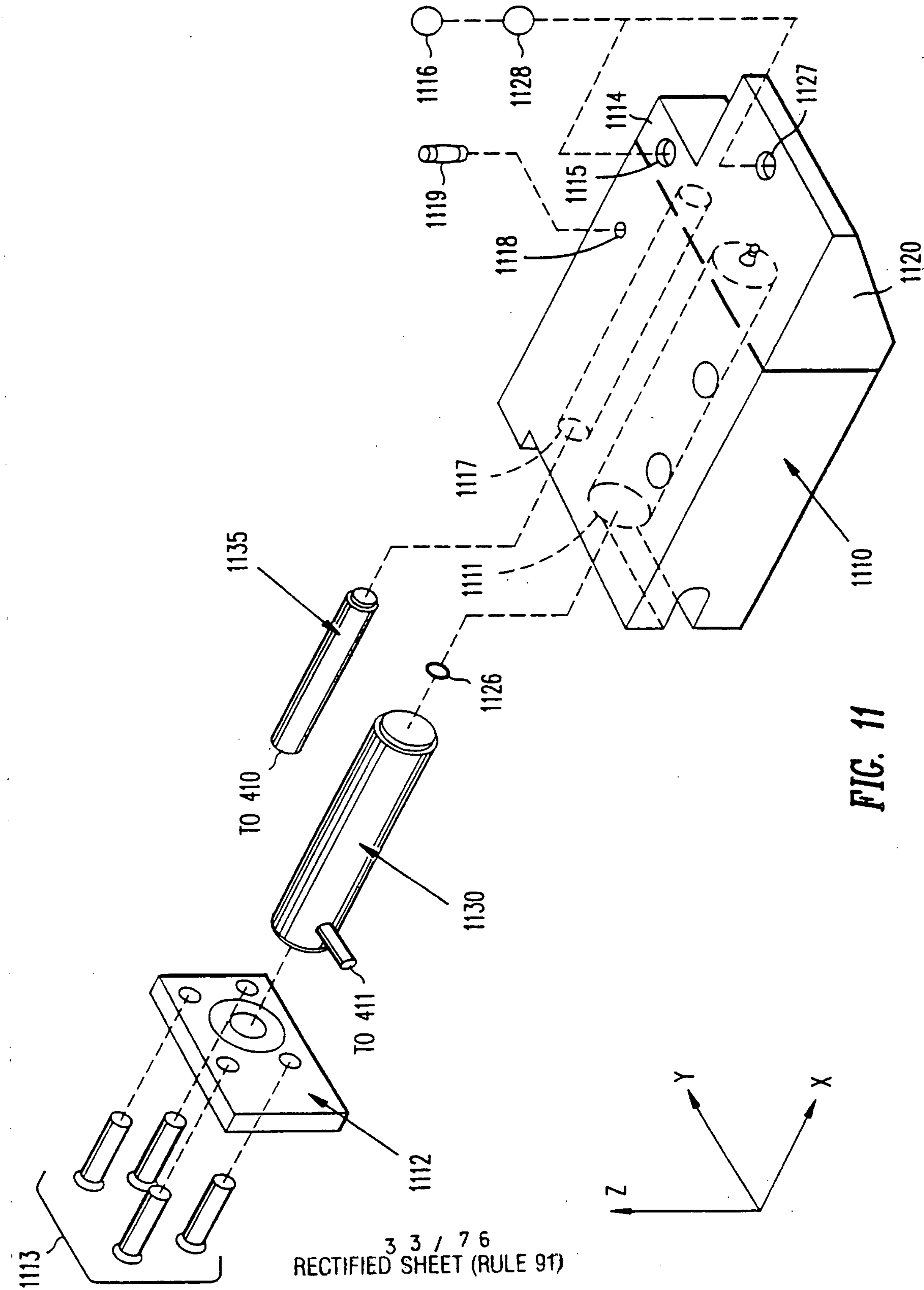

TO 410
TO 411
Y
X
Z

FIG. 11

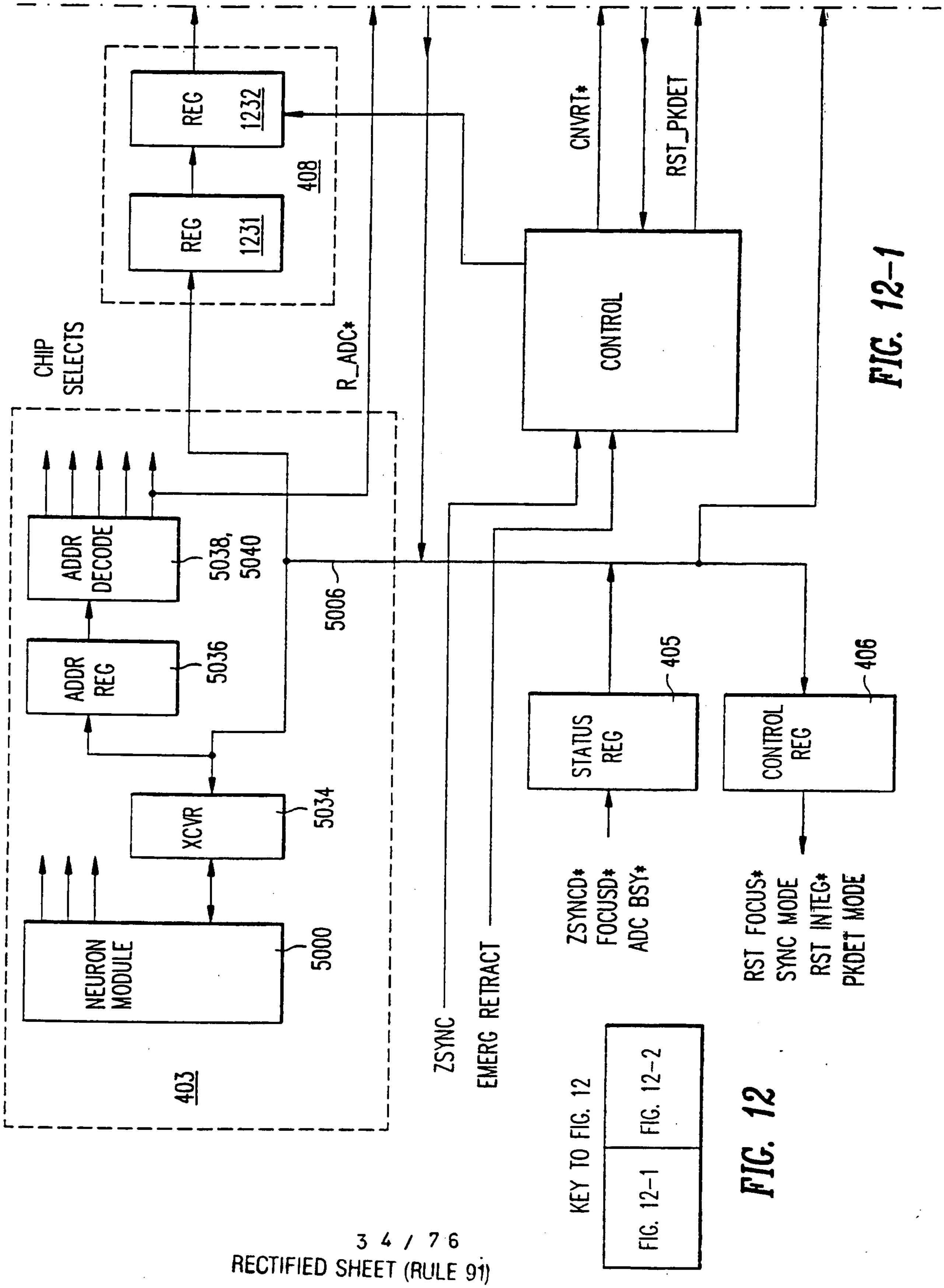

REG
1232
REG
1231
408
CNVRT*
RST_PKDET
CONTROL
CHIP SELECTS
R_ADC*
ADDR DECODE
5038, 5040
ADDR REG
5036
5006
XCVR
5034
NEURON MODULE
5000
403
STATUS REG
405
CONTROL REG
406
ZSYNCD*
FOCUSD*
ADC BSY*
RST FOCUS*
SYNC MODE
RST INTEG*
PKDET* MODE
ZSYNC
EMERG RETRACT
KEY TO FIG. 12
FIG. 12-1
FIG. 12-2

FIG. 12-1

FIG. 12

RECTIFIED SHEET (RULE 91)

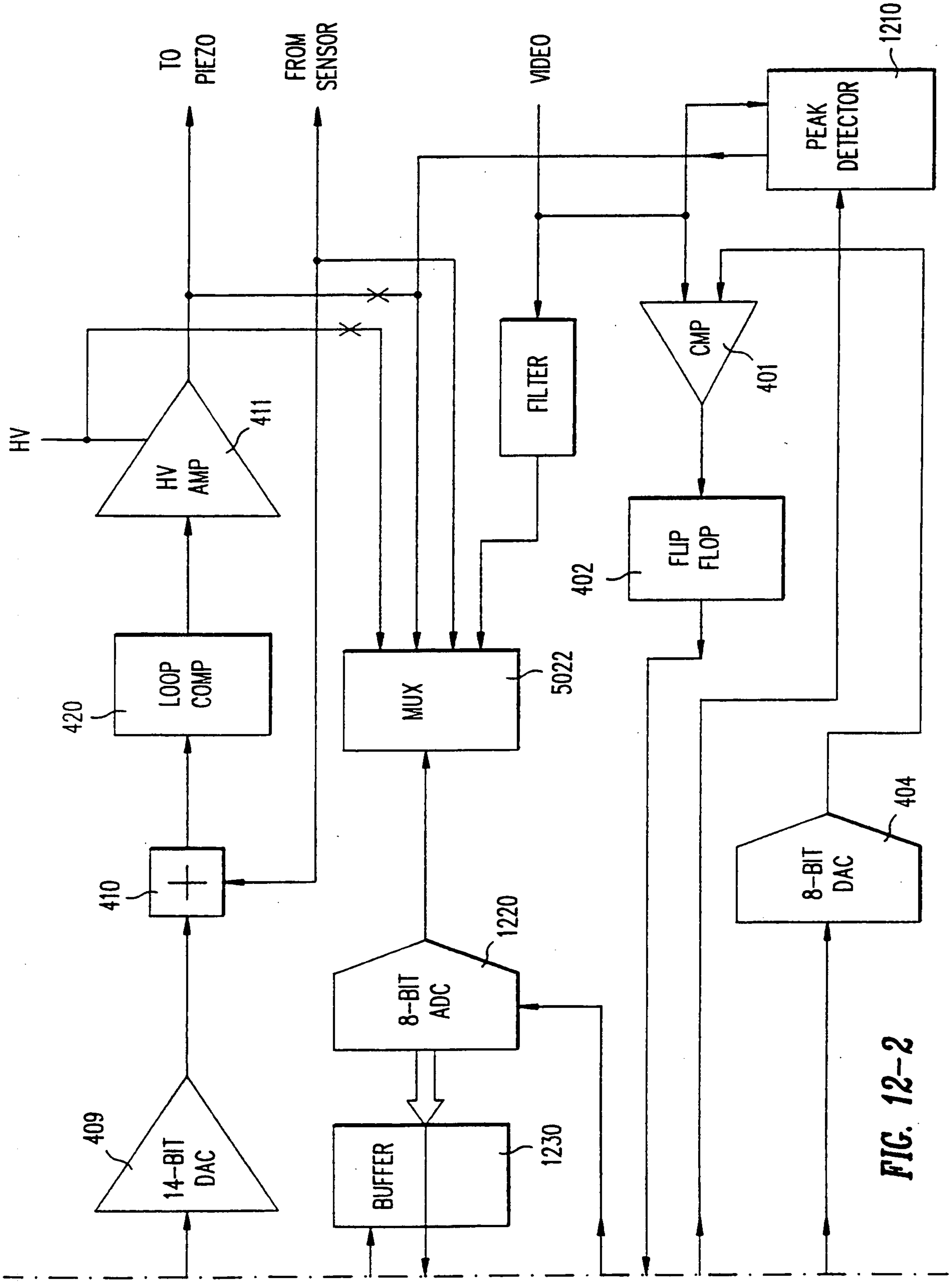

TO PIEZO
FROM SENSOR
VIDEO
1210
PEAK DETECTOR
HV
HV AMP
411
FILTER
CMP
401
420
LOOP COMP
MUX
5022
402
FLIP FLOP
410
1220
8-BIT ADC
404
8-BIT DAC
409
14-BIT DAC
BUFFER
1230

FIG. 12-2

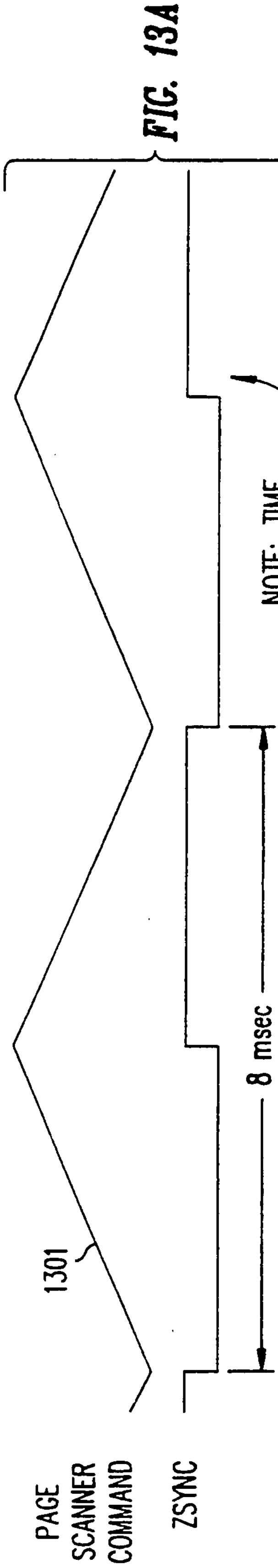

PAGE SCANNER COMMAND
ZSYNC
1301
8 msec

FIG. 13A

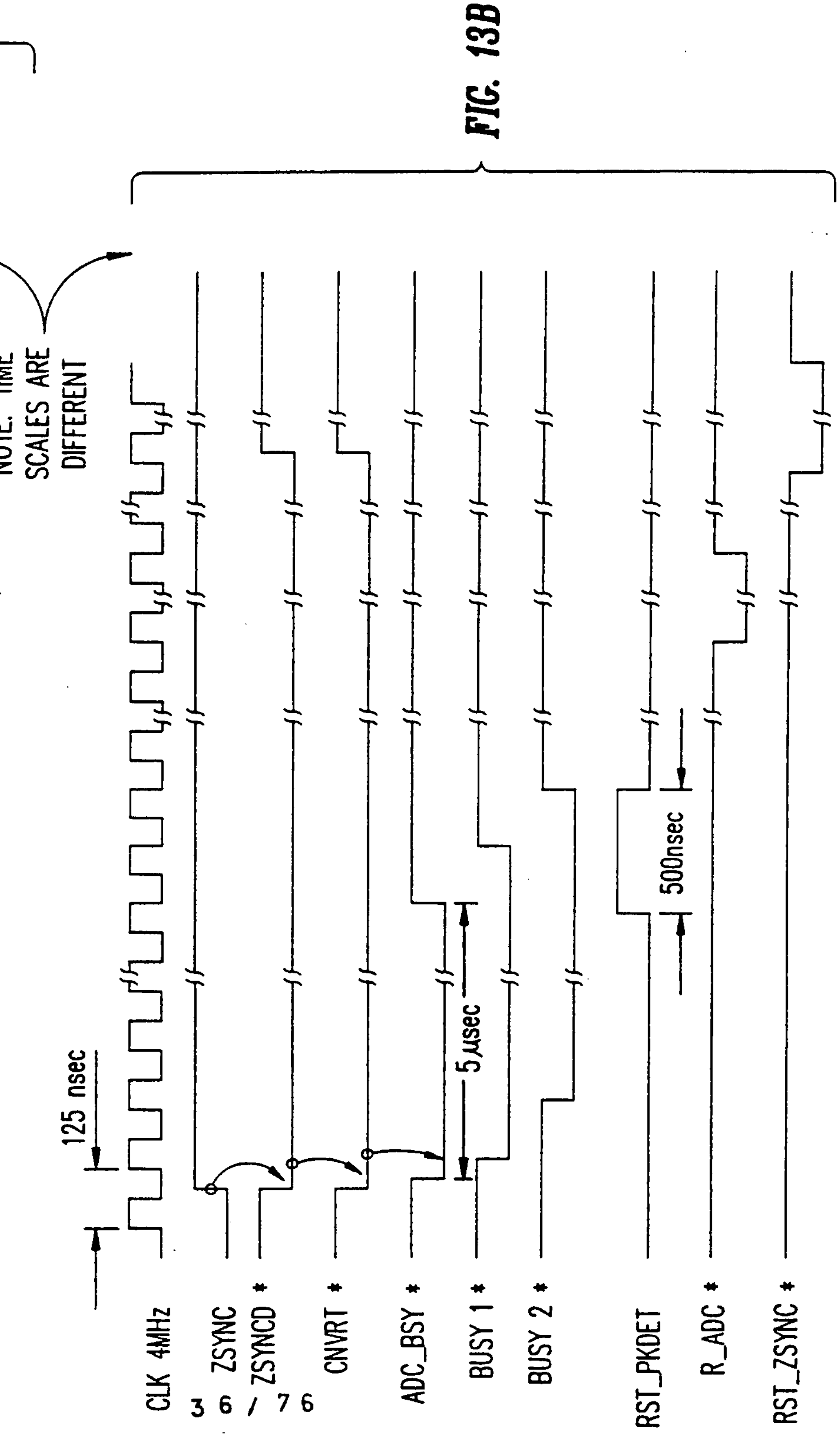

NOTE: TIME SCALES ARE DIFFERENT
125 nsec
5 μsec
500nsec
CLK 4MHz
ZSYNC
ZSYNCD *
CNVRT *
ADC_BSY *
BUSY 1 *
BUSY 2 *
RST_PKDET
R_ADC *
RST_ZSYNC *

FIG. 13B

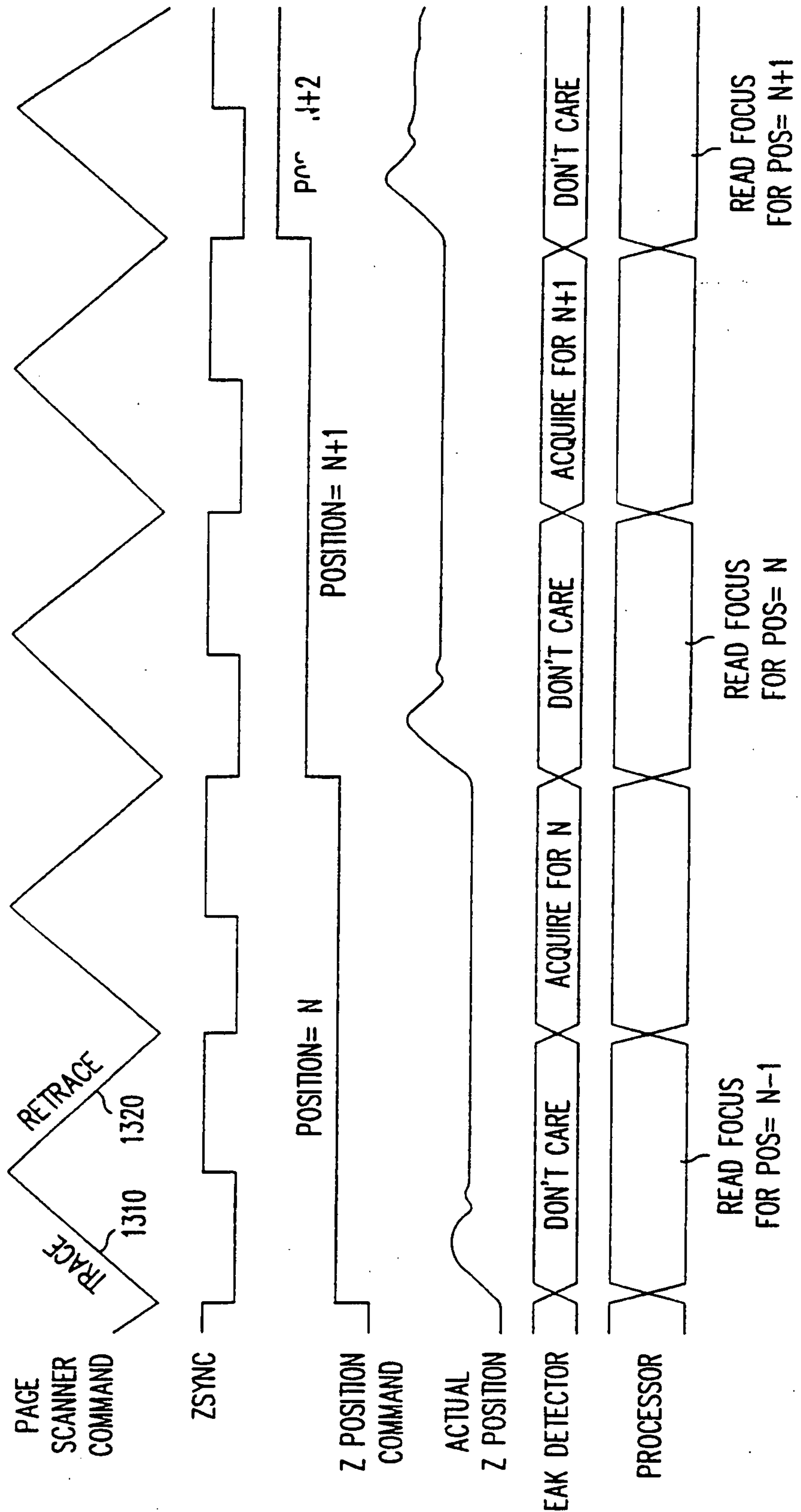
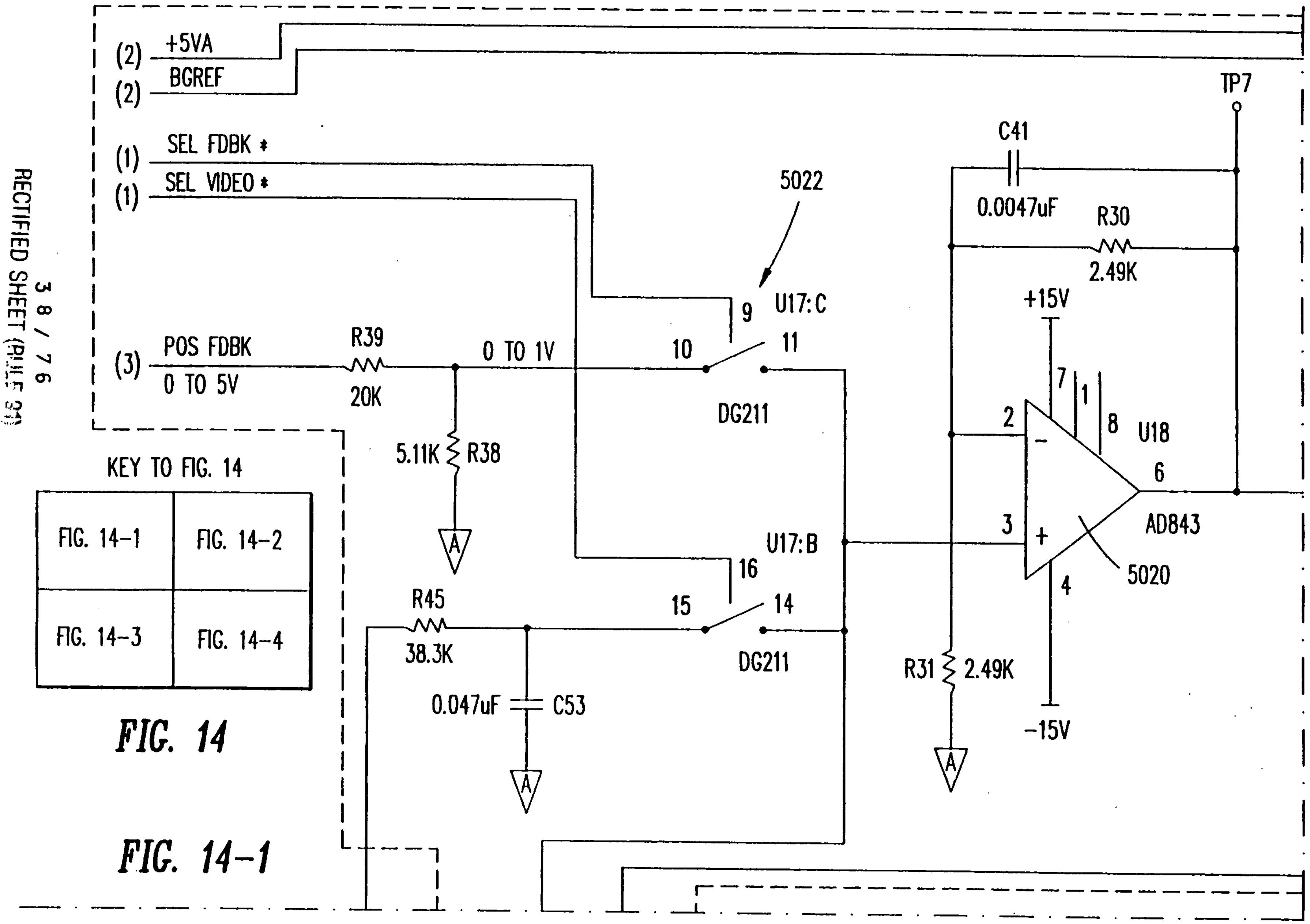

PAGE SCANNER COMMAND
TRACE
1310
RETRACE
1320
ZSYNC
Z POSITION COMMAND
POSITION= N
POSITION= N+1
POS N+2
ACTUAL Z POSITION
PEAK DETECTOR
DON'T CARE
ACQUIRE FOR N
DON'T CARE
ACQUIRE FOR N+1
DON'T CARE
PROCESSOR
READ FOCUS FOR POS= N-1
READ FOCUS FOR POS= N
READ FOCUS FOR POS= N+1

FIG. 13C


(2) +5VA
(2) BGREF
(1) SEL FDBK *
(1) SEL VIDEO *
(3) POS FDBK
0 TO 5V
R39
20K
0 TO 1V
5.11K
R38
A
R45
38.3K
0.047uF
C53
A
5022
9
U17:C
10
11
DG211
U17:B
16
15
14
DG211
C41
0.0047uF
R30
2.49K
TP7
+15V
7
1
8
U18
2
-
6
3
+
AD843
4
5020
R31
2.49K
A
-15V

KEY TO FIG. 14

FIG. 14-1	FIG. 14-2
FIG. 14-3	FIG. 14-4

FIG. 14

FIG. 14-1

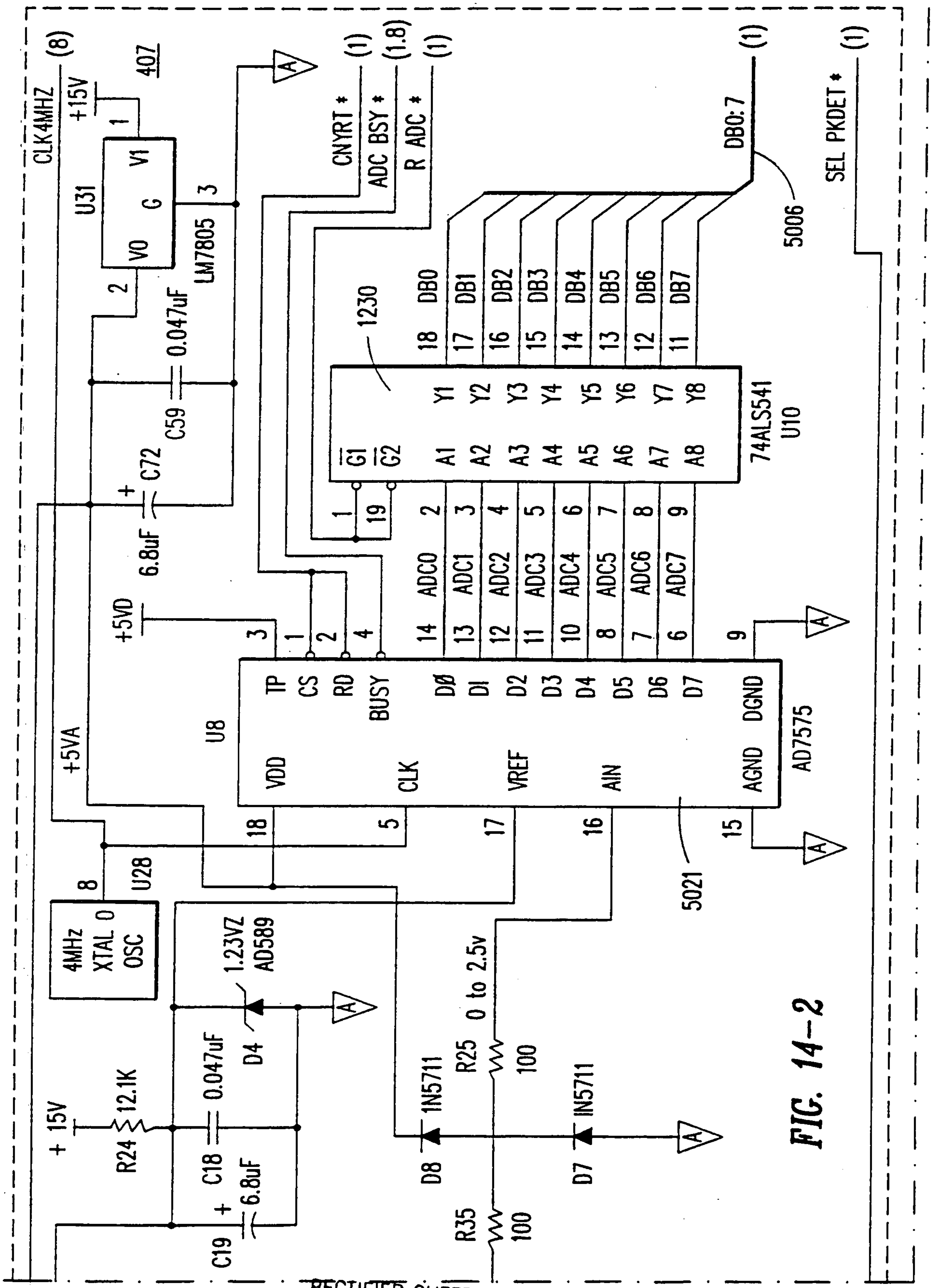

CLK4MHZ
(8)
+15V
407
U31
V1
G
V0
LM7805
1
2
3
0.047uF
C59
C72
6.8uF
A
CNYRT *
ADC BSY *
R ADC *
(1)
(1,8)
(1)
DB0:7
(1)
5006
SEL PKDET *
(1)
1230
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
18
17
16
15
14
13
12
11
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
G1
G2
A1
A2
A3
A4
A5
A6
A7
A8
74ALS541
U10
19
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
ADC6
ADC7
+5VD
TP
CS
RD
BUSY
DØ
DI
D2
D3
D4
D5
D6
D7
DGND
AGND
VDD
CLK
VREF
AIN
U8
AD7575
+5VA
5021
U28
8
4MHz
XTAL 0
OSC
1.23VZ
AD589
D4
0 to 2.5v
R25
100
1N5711
D8
D7
+ 15V
R24
12.1K
0.047uF
C18
6.8uF
C19
R35
100

FIG. 14-2

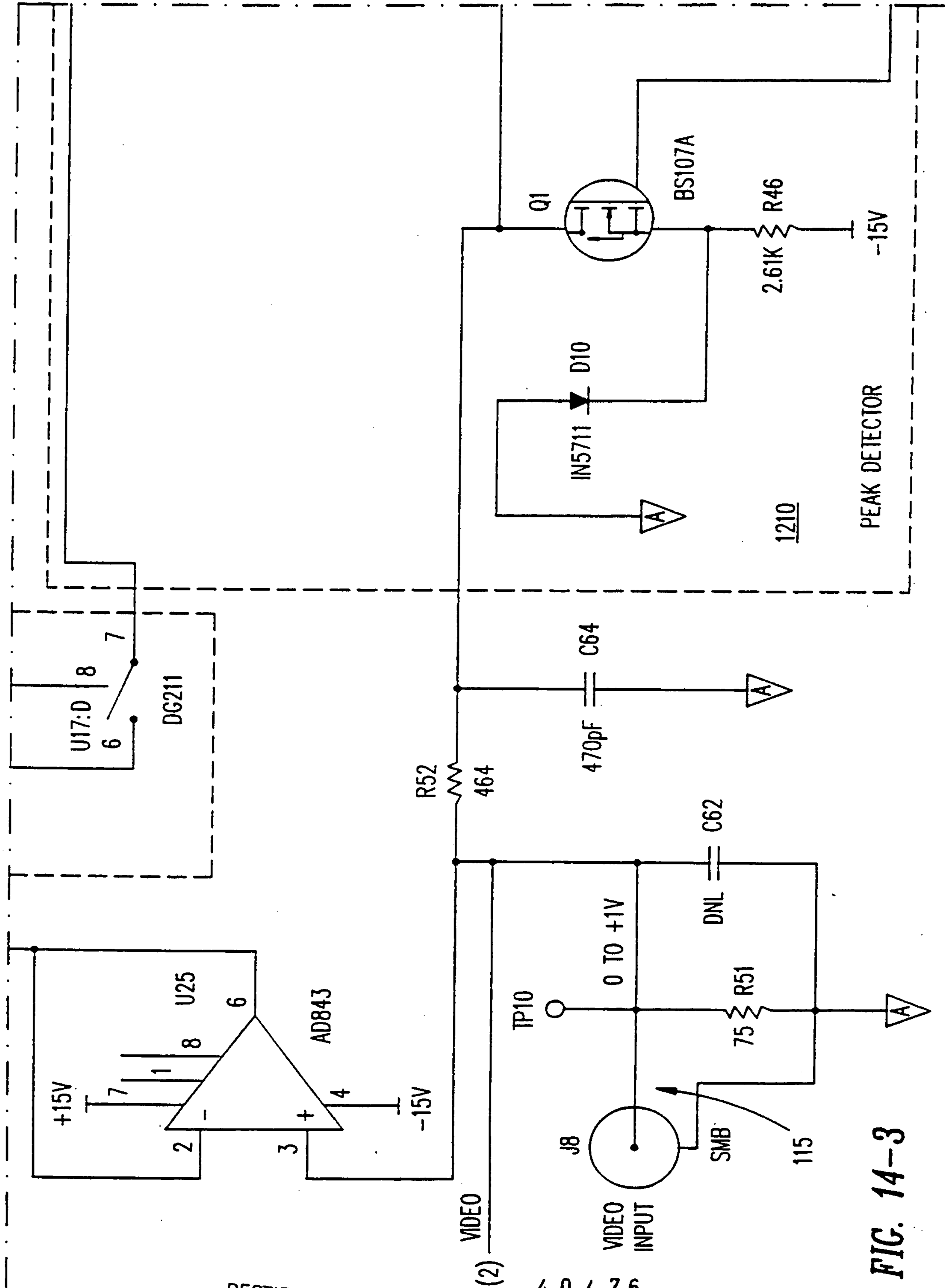

U25
AD843
+15V
-15V
-15V
2
3
4
6
7
1
8
-
+
U17:D
DG211
6
8
7
R52
464
C64
470pF
C62
DNL
VIDEO
(2)
TP10
0 TO +1V
R51
75
J8
VIDEO INPUT
SMB
115
A
A
A
Q1
BS107A
R46
2.61K
D10
IN5711
1210
PEAK DETECTOR

FIG. 14-3

RECTIFIED SHEET (RULE 91)

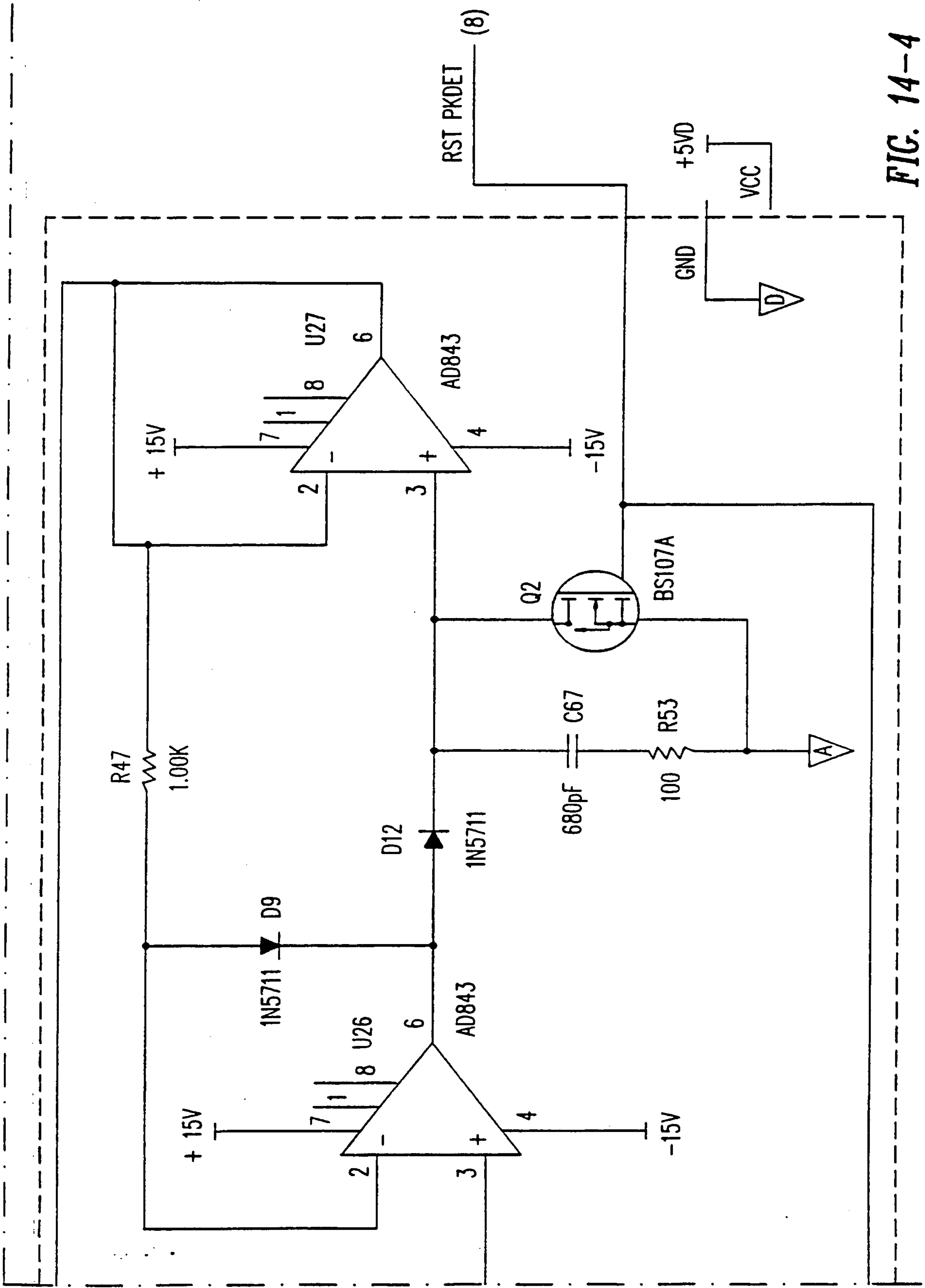
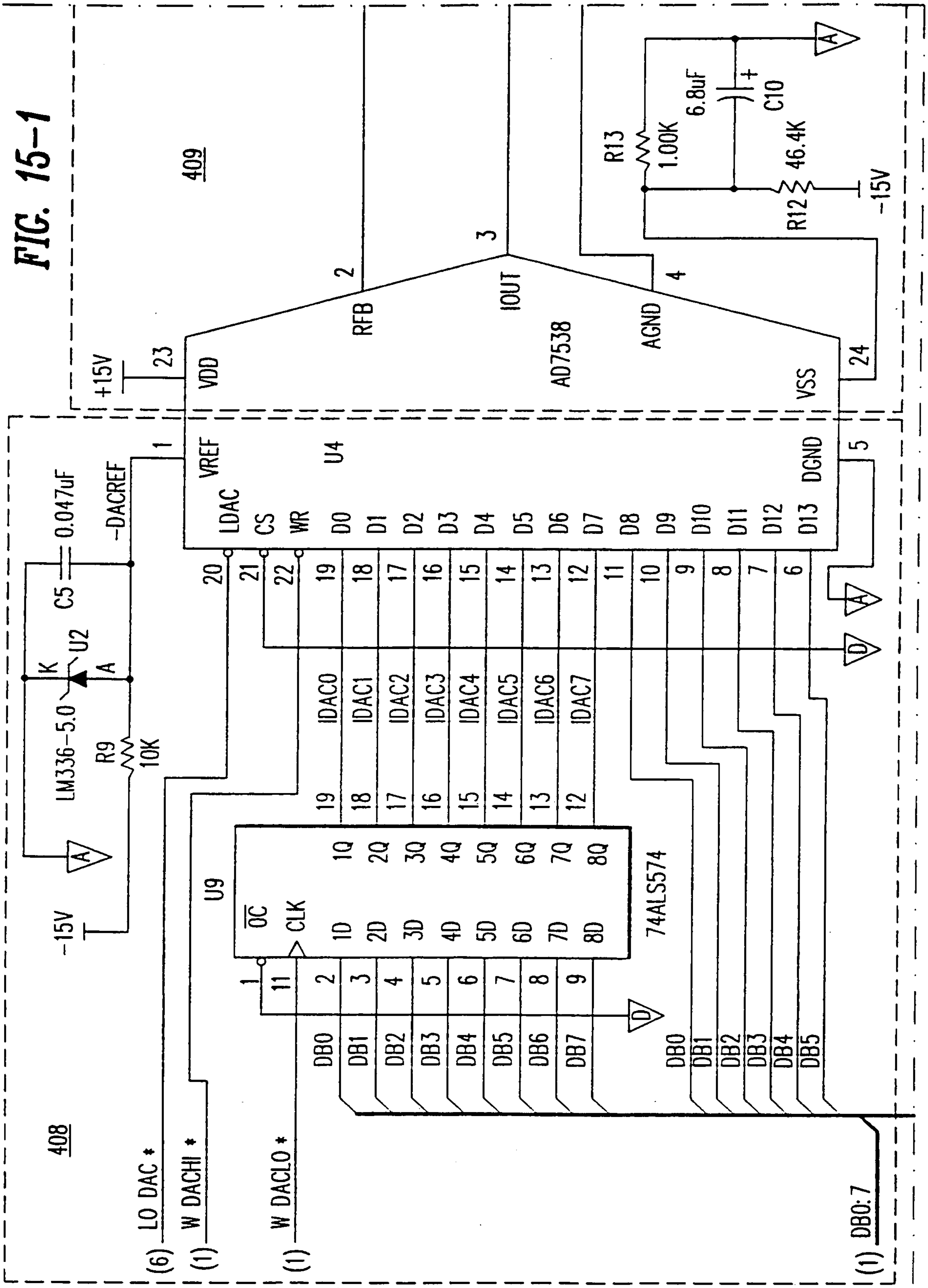

RST PKDET
(8)
+5VD
VCC
GND
D
U27
6
8
1
7
+ 15V
AD843
4
-15V
2
3
-
+
Q2
BS107A
C67
680pF
R53
100
A
R47
1.00K
D12
1N5711
D9
1N5711
U26
6
AD843
8
1
7
+ 15V
4
-15V
2
3

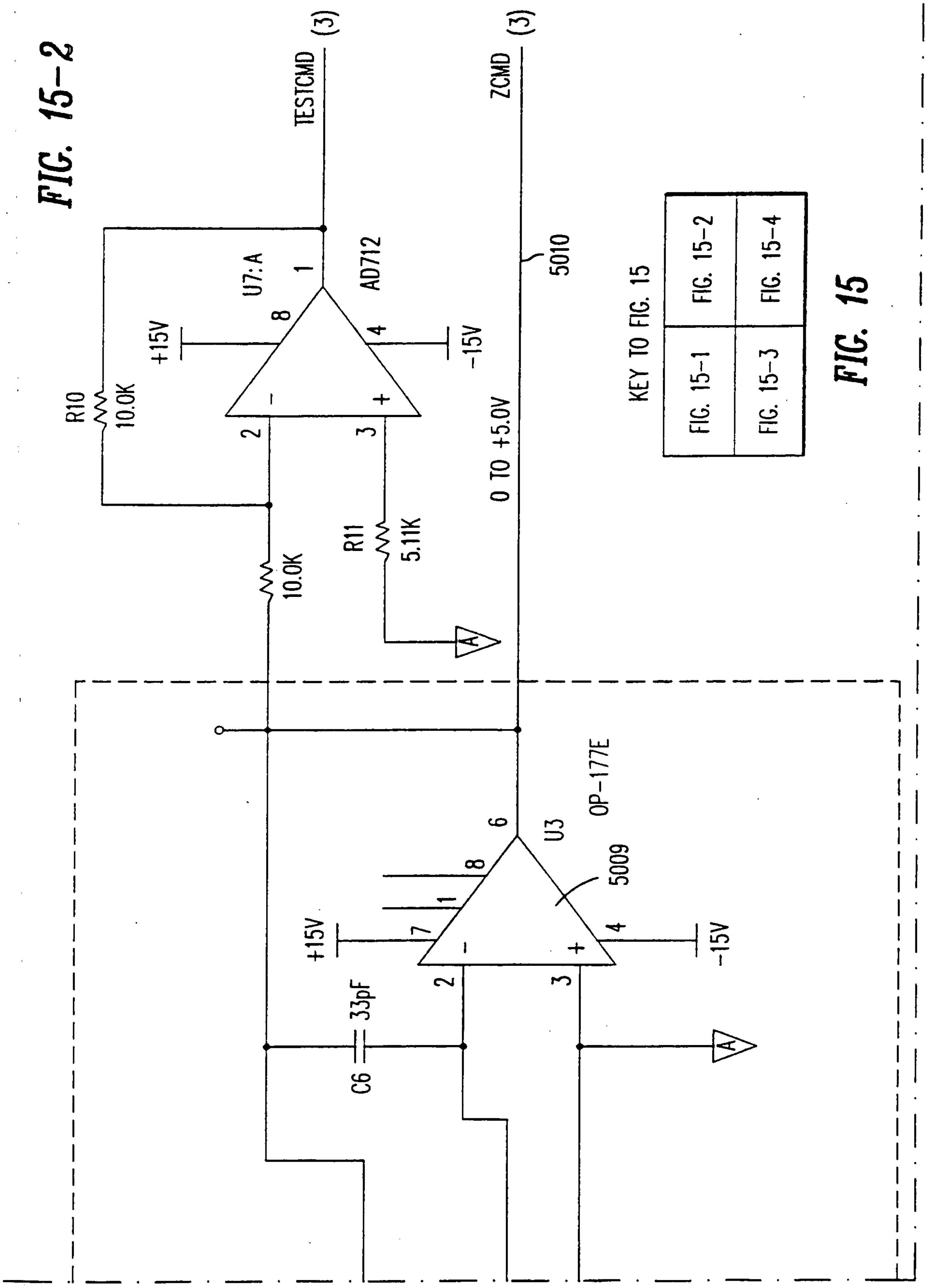
FIG. 14-4

FIG. 15-1


409
408
+15V
23
VDD
RFB
2
IOUT
3
AD7538
AGND
4
VSS
24
R13
1.00K
6.8uF
C10
46.4K
R12
-15V
A
VREF
1
U4
DGND
5
LDAC
CS
WR
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
20
21
22
19
18
17
16
15
14
13
12
11
10
9
8
7
6
A
D
0.047uF
C5
-DACREF
K
U2
A
LM336-5.0
R9
10K
A
-15V
IDAC0
IDAC1
IDAC2
IDAC3
IDAC4
IDAC5
IDAC6
IDAC7
U9
OC
CLK
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
1D
2D
3D
4D
5D
6D
7D
8D
74ALS574
19
18
17
16
15
14
13
12
1
11
2
3
4
5
6
7
8
9
D
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DB0
DB1
DB2
DB3
DB4
DB5
(6) LO DAC *
(1) W DACHI *
(1) W DACLO *
(1) DB0:7

RECTIFIED SHEET (RULE 91)

FIG. 15-2


TESTCMD
(3)
ZCMD
(3)
R10
10.0K
+15V
-15V
U7:A
AD712
8
4
1
2
3
R11
5.11K
10.0K
0 TO +5.0V
5010
A
OP-177E
U3
5009
6
8
1
7
4
2
3
+15V
-15V
C6
33pF
A

KEY TO FIG. 15

FIG. 15-1	FIG. 15-2
FIG. 15-3	FIG. 15-4

FIG. 15

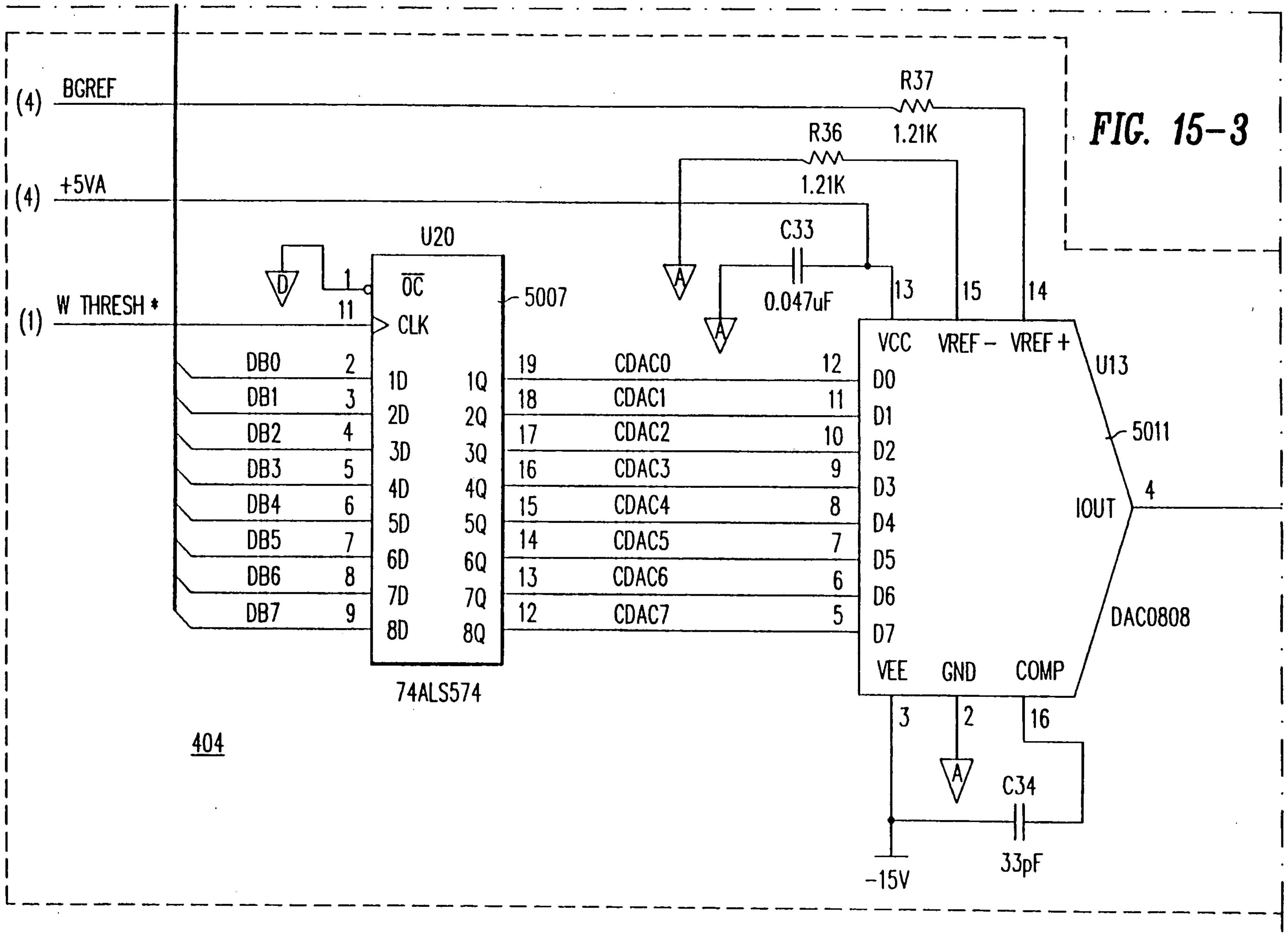

(4)
BGREF
(4)
+5VA
(1)
W THRESH *
R37
1.21K
R36
1.21K
C33
0.047uF
A
A
A
D
U20
1
11
OC
CLK
5007
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
2
3
4
5
6
7
8
9
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
74ALS574
19
18
17
16
15
14
13
12
CDAC0
CDAC1
CDAC2
CDAC3
CDAC4
CDAC5
CDAC6
CDAC7
12
11
10
9
8
7
6
5
13
15
14
VCC
VREF –
VREF +
D0
D1
D2
D3
D4
D5
D6
D7
VEE
GND
COMP
U13
5011
IOUT
4
DAC0808
3
2
16
C34
33pF
–15V
404

FIG. 15-3

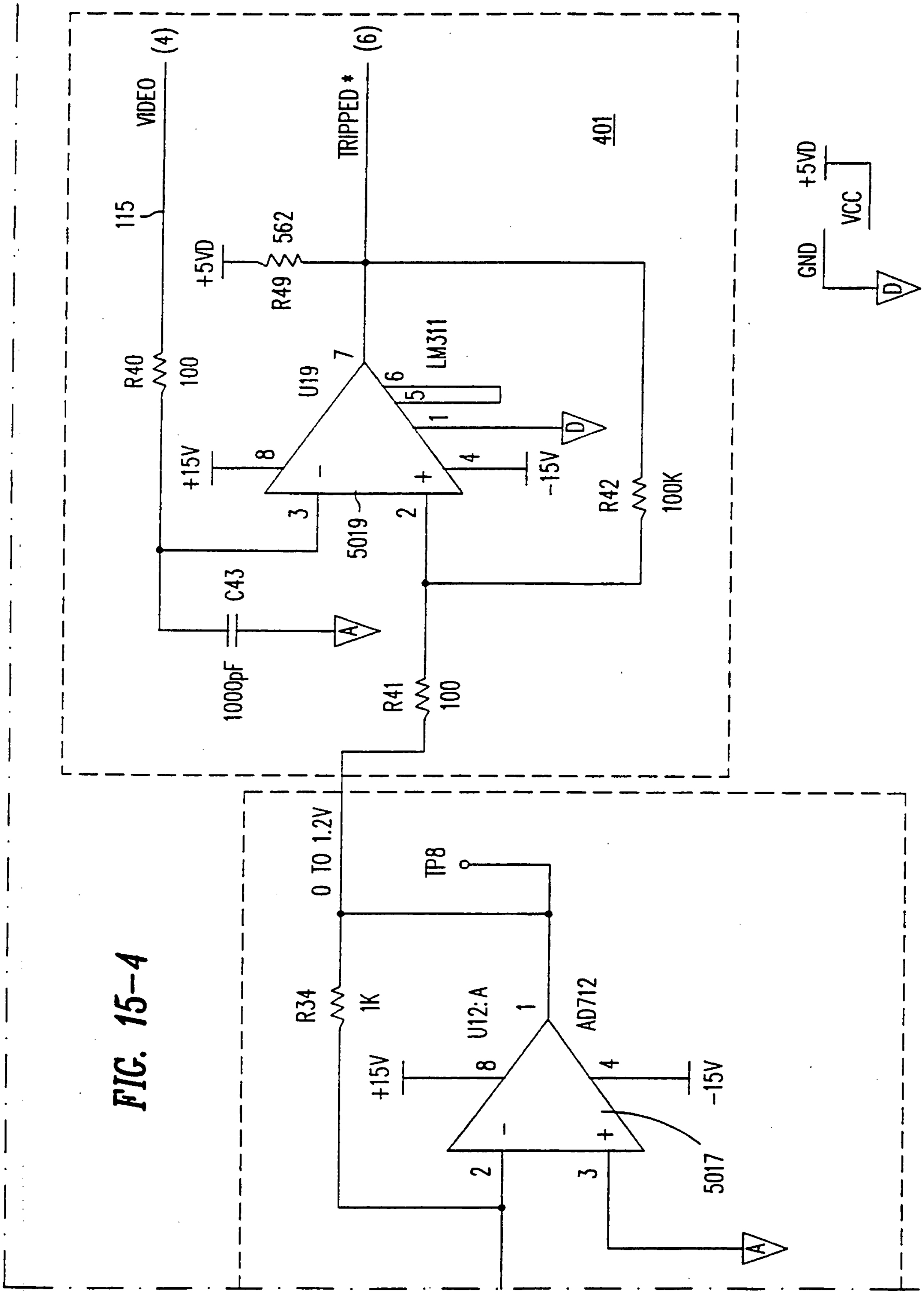
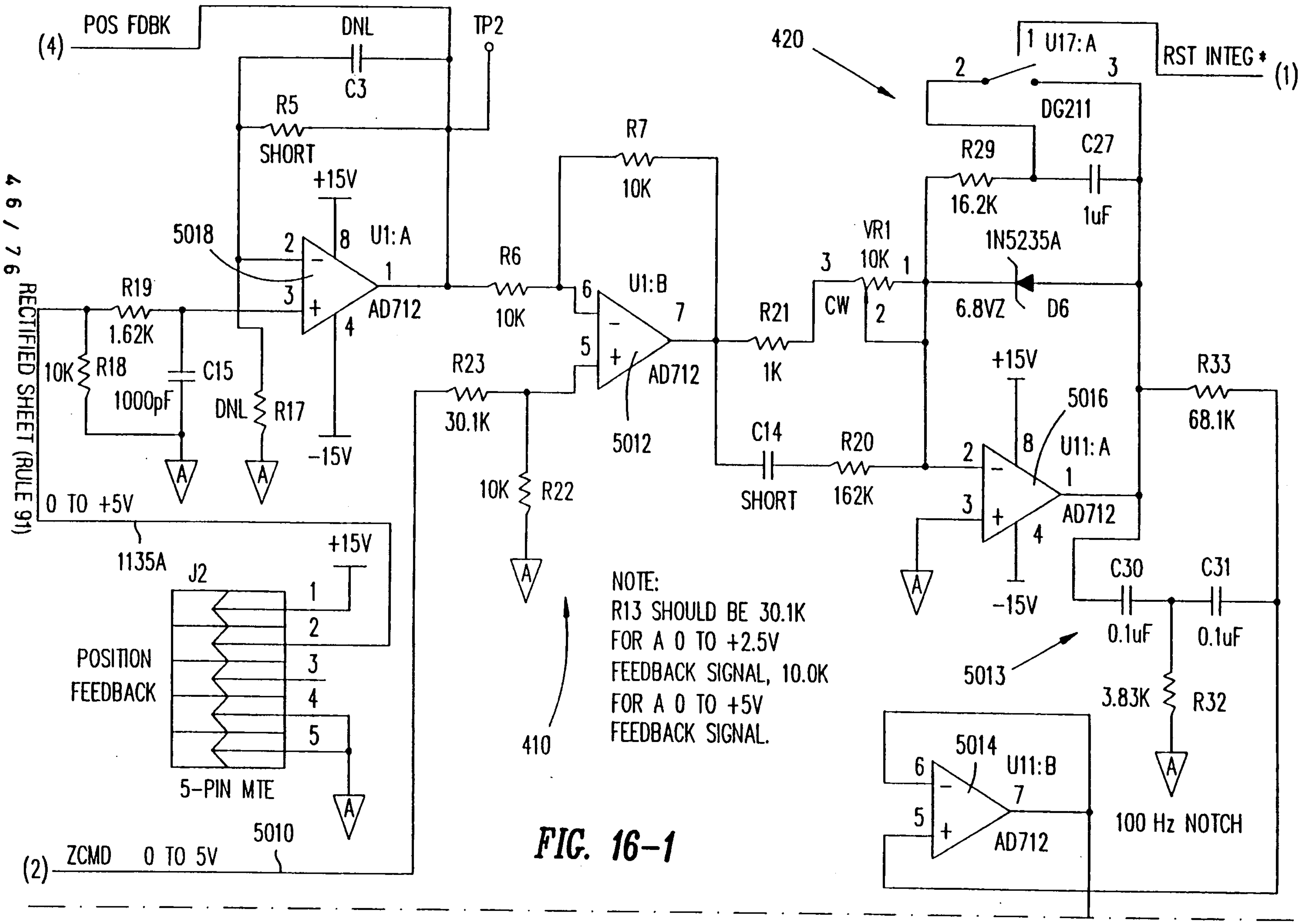

VIDEO
(4)
115
R40
100
TRIPPED *
(6)
+5VD
R49
562
401
U19
LM311
7
6
5
1
8
4
3
2
+15V
-15V
5019
R42
100K
C43
1000pF
R41
100
+5VD
VCC
GND
D
A
0 TO 1.2V
TP8
R34
1K
U12:A
AD712
1
8
4
2
3
+15V
-15V
5017

FIG. 15-4


(4)
POS FDBK
DNL
C3
TP2
R5
SHORT
+15V
5018
2
8
U1:A
3
1
AD712
4
−15V
R19
1.62K
10K
R18
C15
1000pF
DNL
R17
A
0 TO +5V
1135A
J2
+15V
1
2
3
4
5
POSITION
FEEDBACK
5-PIN MTE
5010
(2)
ZCMD
0 TO 5V
R6
10K
R7
10K
6
U1:B
7
5
AD712
5012
R23
30.1K
10K
R22
410
R21
1K
C14
SHORT
R20
162K
3
VR1
10K
1
CW
2
420
U17:A
2
3
DG211
RST INTEG *
(1)
R29
16.2K
C27
1uF
1N5235A
6.8VZ
D6
+15V
5016
8
U11:A
1
AD712
4
−15V
R33
68.1K
C30
C31
0.1uF
0.1uF
3.83K
R32
5013
5014
U11:B
6
7
5
AD712
100 Hz NOTCH

NOTE:
R13 SHOULD BE 30.1K
FOR A 0 TO +2.5V
FEEDBACK SIGNAL, 10.0K
FOR A 0 TO +5V
FEEDBACK SIGNAL.

FIG. 16–1

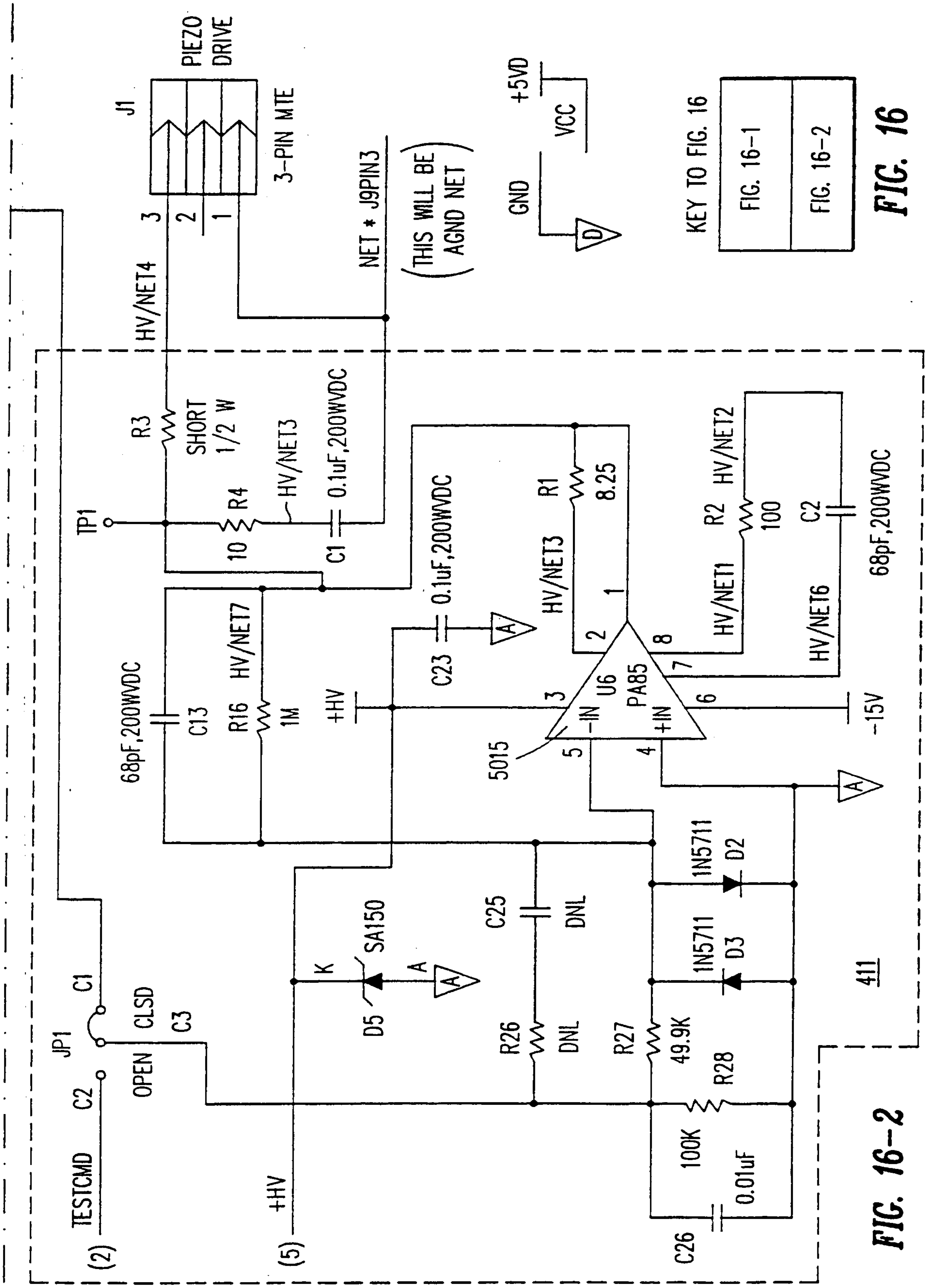
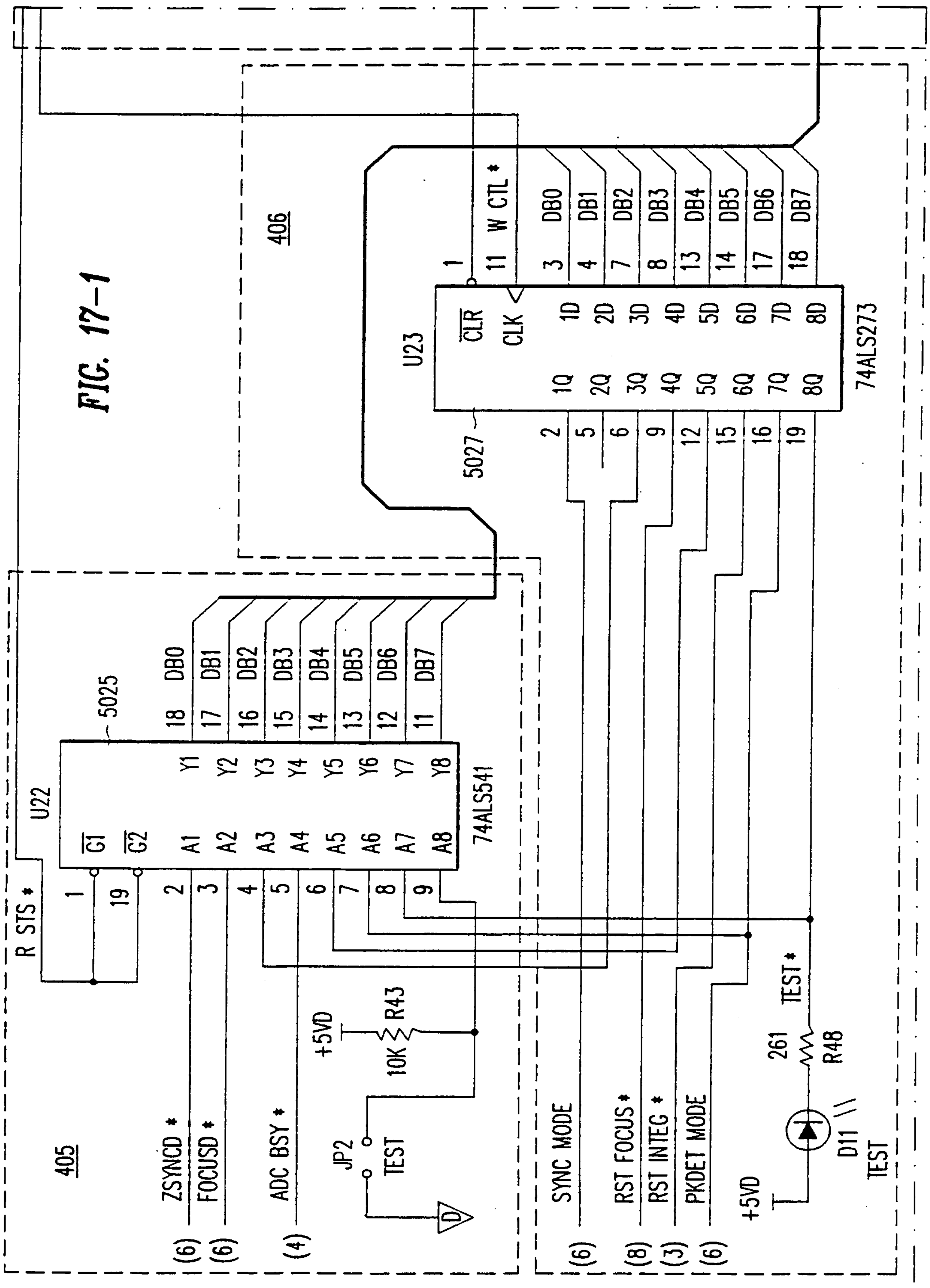

PIEZO DRIVE
J1
3-PIN MTE
3
2
1
HV/NET4
NET * J9PIN3
(THIS WILL BE AGND NET)
+5VD
VCC
GND
D
KEY TO FIG. 16
FIG. 16-1
FIG. 16-2
R3
SHORT
1/2 W
TP1
R4
10
HV/NET3
C1
0.1uF,200WVDC
R1
8.25
HV/NET2
R2
100
C2
68pF,200WVDC
HV/NET1
HV/NET6
HV/NET3
HV/NET7
C23
0.1uF,200WVDC
A
68pF,200WVDC
C13
R16
1M
+HV
U6
PA85
-IN
+IN
1
2
3
4
5
6
7
8
-15V
5015
A
1N5711
D2
1N5711
D3
C25
DNL
K
SA150
A
D5
A
411
JP1
C1
CLSD
C3
OPEN
C2
R26
DNL
R27
49.9K
R28
100K
0.01uF
C26
TESTCMD
(2)
+HV
(5)

FIG. 16-2

FIG. 16

FIG. 17-1


406
U23
74ALS273
5027
CLR
CLK
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
1
11
3
4
7
8
13
14
17
18
2
5
6
9
12
15
16
19
W CTL *
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
405
U22
74ALS541
5025
G1
G2
A1
A2
A3
A4
A5
A6
A7
A8
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
R STS *
1
19
2
3
4
5
6
7
8
9
18
17
16
15
14
13
12
11
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
ZSYNCD *
FOCUSD *
ADC BSY *
(6)
(6)
(4)
+5VD
R43
10K
JP2
TEST
D
SYNC MODE
RST FOCUS *
RST INTEG *
PKDET MODE
(6)
(8)
(3)
(6)
TEST *
261
R48
D11
TEST
+5VD

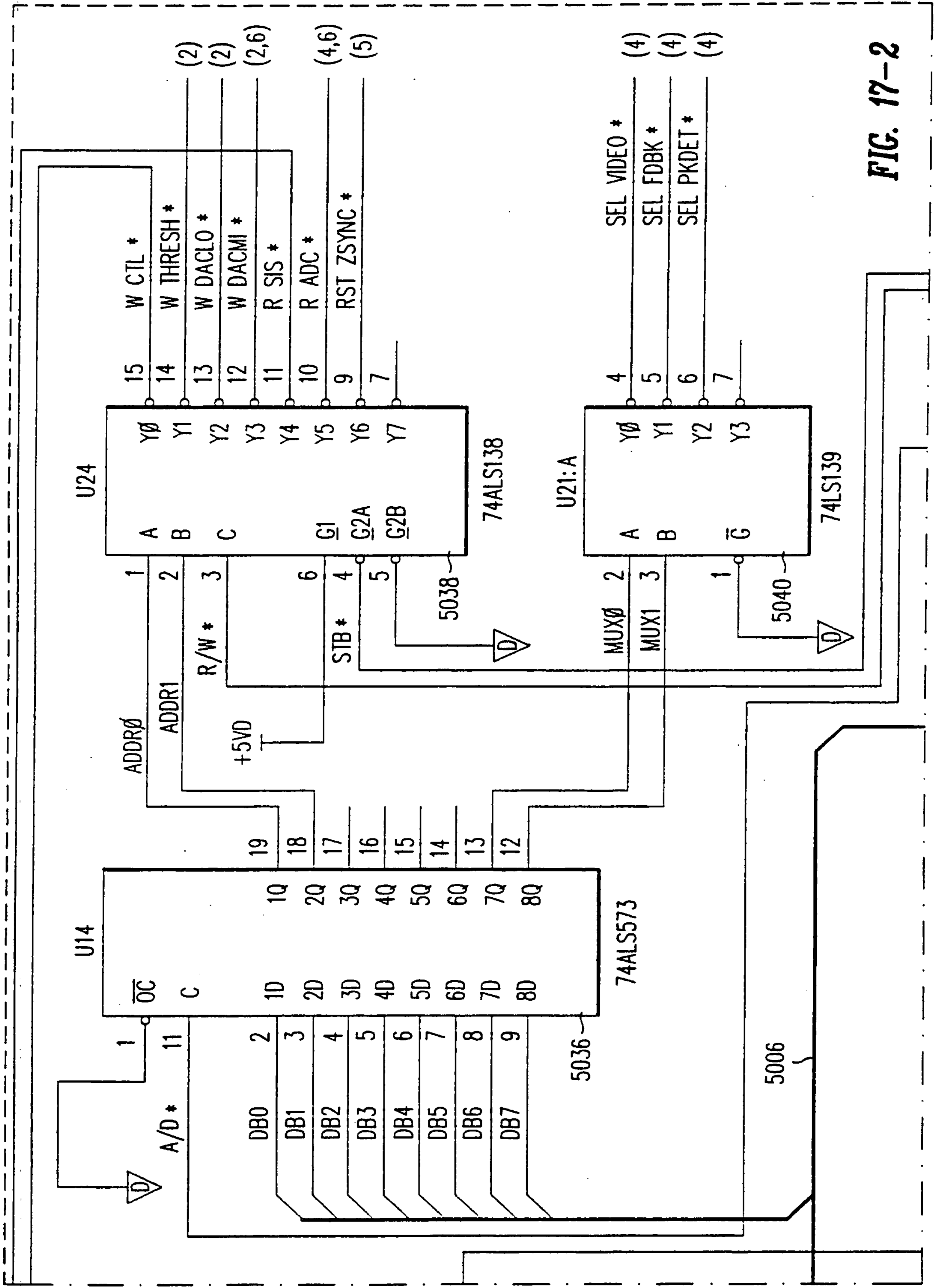

U24
74ALS138
A
B
C
G1
G2A
G2B
YØ
Y1
Y2
Y3
Y4
Y5
Y6
Y7
15
14
13
12
11
10
9
7
1
2
3
6
4
5
5038
W CTL *
W THRESH *
W DACLO *
W DACMI *
R SIS *
R ADC *
RST ZSYNC *
(2)
(2)
(2,6)
(4,6)
(5)
U21:A
74LS139
A
B
G
YØ
Y1
Y2
Y3
4
5
6
7
2
3
1
5040
SEL VIDEO *
SEL FDBK *
SEL PKDET *
(4)
(4)
(4)
MUXØ
MUX1
R/W *
STB *
+5VD
ADDRØ
ADDR1
D
D
U14
74ALS573
OC
C
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
19
18
17
16
15
14
13
12
1
11
2
3
4
5
6
7
8
9
5036
A/D *
D
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
5006

FIG. 17-2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/00665

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G01J 1/20
US CL :250/201.2, 201.3, 204, 216; 359/368, 381
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 250/201.2, 201.3, 204, 216; 359/368, 381

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS
S (CONFOCAL(2A)MICROSCOP?) AND FOCUS? AND FILTER AND (APERTURE? OR PINHOLE? OR OPENING?)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 4,342,905 (FUJII ET AL.) 03 AUGUST 1982, (see entire document)	1-6
Y	US, A, 4,844,617 (KELDERMAN ET AL.) 04 JULY 1989 (see entire document)	1-30
X	US, A, 5,122,648 (COHEN ET AL.) 16 JUNE 1992 (see entire document)	1-10
A,P	US, A, 5,306,902 (GOODMAN) 26 APRIL 1994 (see entire document)	1-32

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

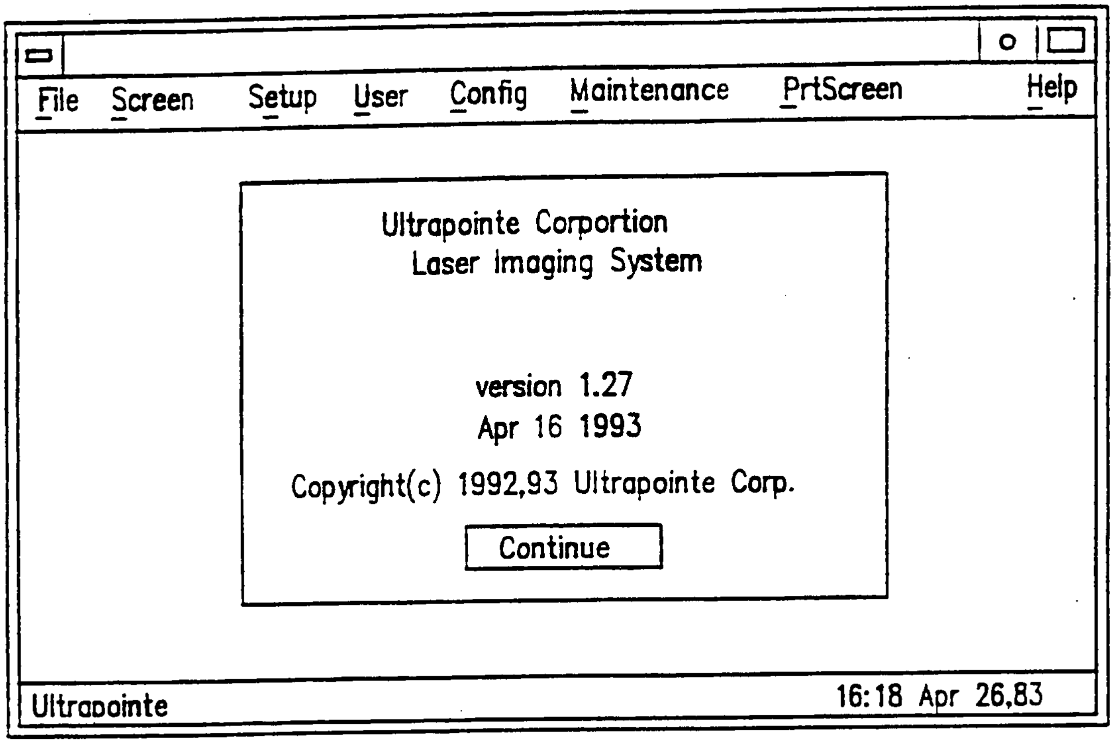
Date of the actual completion of the international search
27 APRIL 1995

Date of mailing of the international search report
03 MAY 1995

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
STEPHONE B. ALLEN
Telephone No. (703) 305-4828

Form PCT/ISA/210 (second sheet)(July 1992)★


File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
Continue

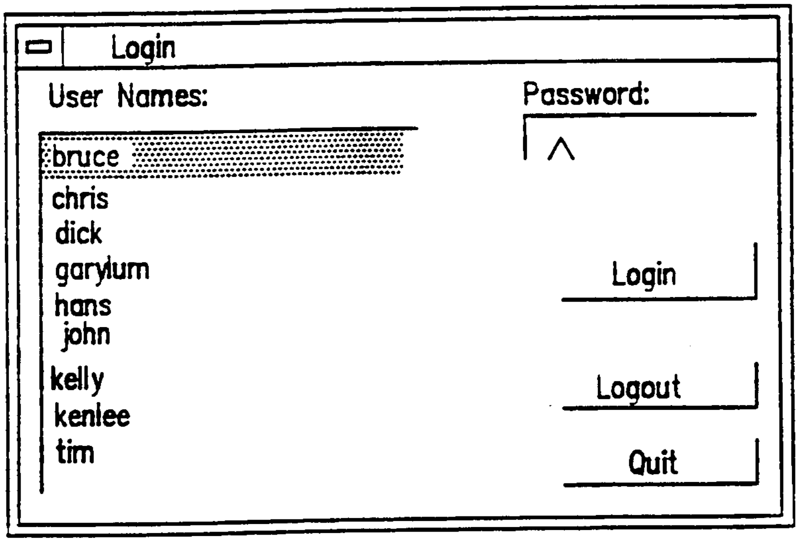
Ultrapointe Corportion
Laser Imaging System

version 1.27
Apr 16 1993

Copyright(c) 1992,93 Ultrapointe Corp.

Ultrapointe
16:18 Apr 26,83

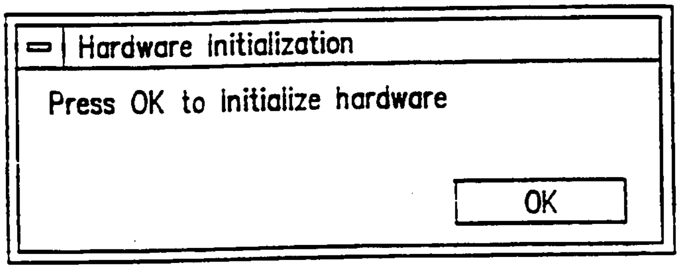
FIG. 45


Login
Login
Logout
Quit

User Names:
bruce
chris
dick
garylum
hans
john
kelly
kenlee
tim

Password:

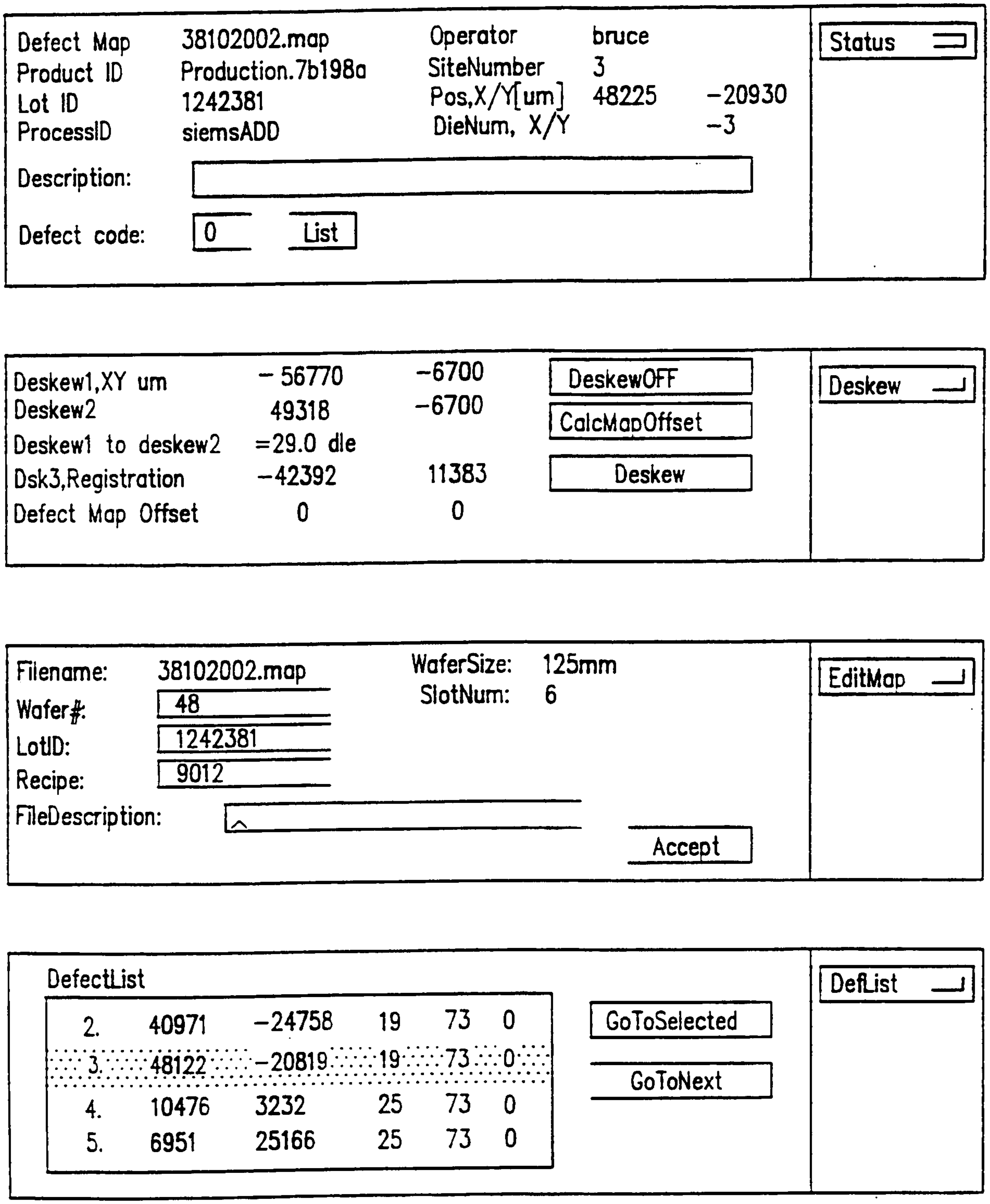
FIG. 46



Hardware Initialization

Press OK to initialize hardware

FIG. 47



Defect Map 38102002.map
Product ID Production.7b198a
Lot ID 1242381
ProcessID siemsADD
Operator bruce
SiteNumber 3
Pos,X/Y[um] 48225 -20930
DieNum, X/Y -3
Description:
Defect code: 0

Deskew1,XY um - 56770 -6700
Deskew2 49318 -6700
Deskew1 to deskew2 =29.0 dle
Dsk3,Registration -42392 11383
Defect Map Offset 0 0

Filename: 38102002.map
Wafer#: 48
LotID: 1242381
Recipe: 9012
FileDescription:
WaferSize: 125mm
SlotNum: 6

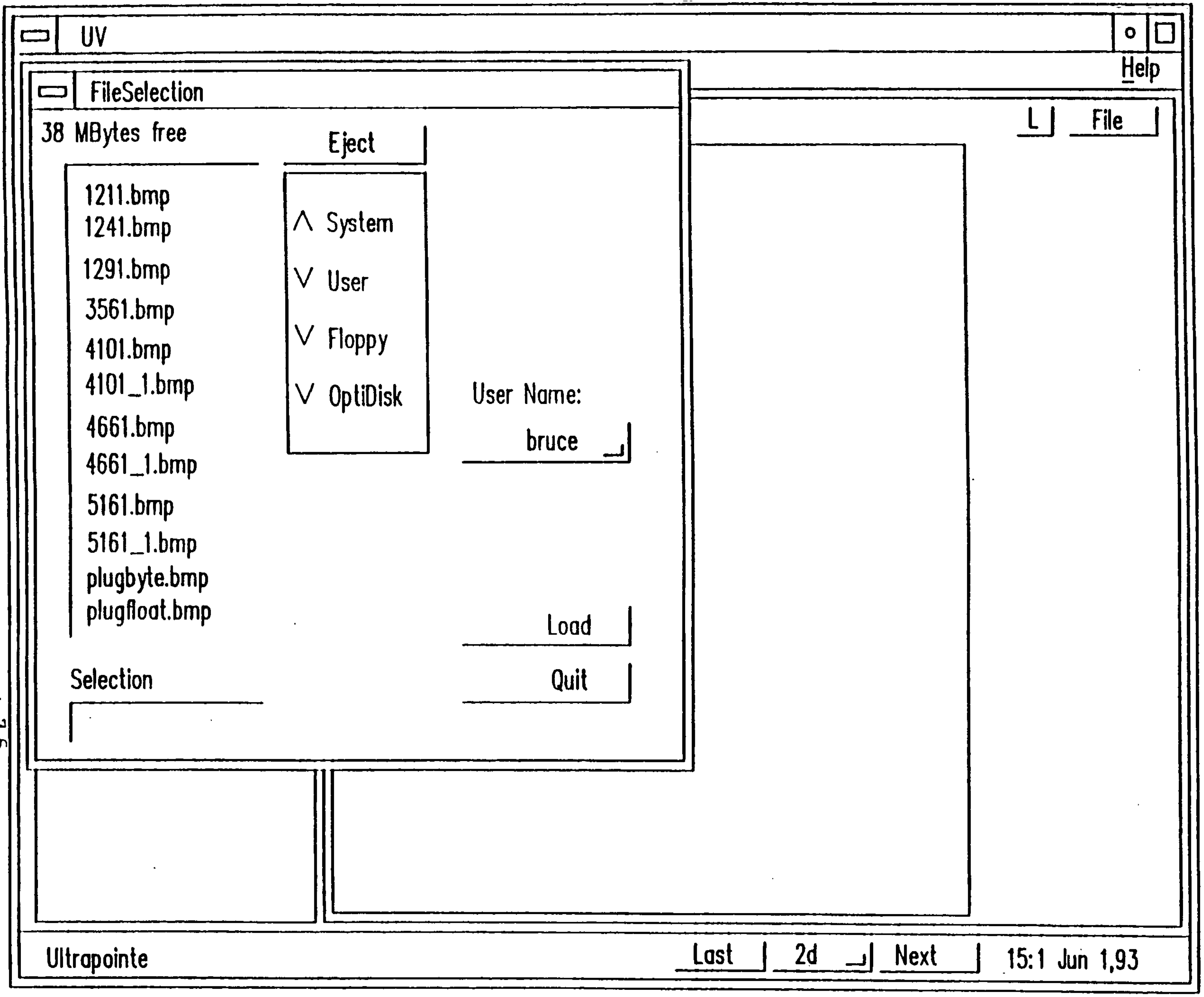
DefectList

2.	40971	-24758	19	73	0
3.	48122	-20819	19	73	0
4.	10476	3232	25	73	0
5.	6951	25166	25	73	0

Status
List
DeskewOFF
CalcMapOffset
Deskew
Deskew
EditMap
Accept
DefList
GoToSelected
GoToNext

Text and Status Windows

FIG. 44


UV
Help
L
File
FileSelection
38 MBytes free
Eject
1211.bmp
1241.bmp
1291.bmp
3561.bmp
4101.bmp
4101_1.bmp
4661.bmp
4661_1.bmp
5161.bmp
5161_1.bmp
plugbyte.bmp
plugfloat.bmp
∧ System
∨ User
∨ Floppy
∨ OptiDisk
Load
Quit
Selection
Ultrapointe
Last
2d
Next
15:1 Jun 1,93

User Name: bruce

Library Window

FIG. 43

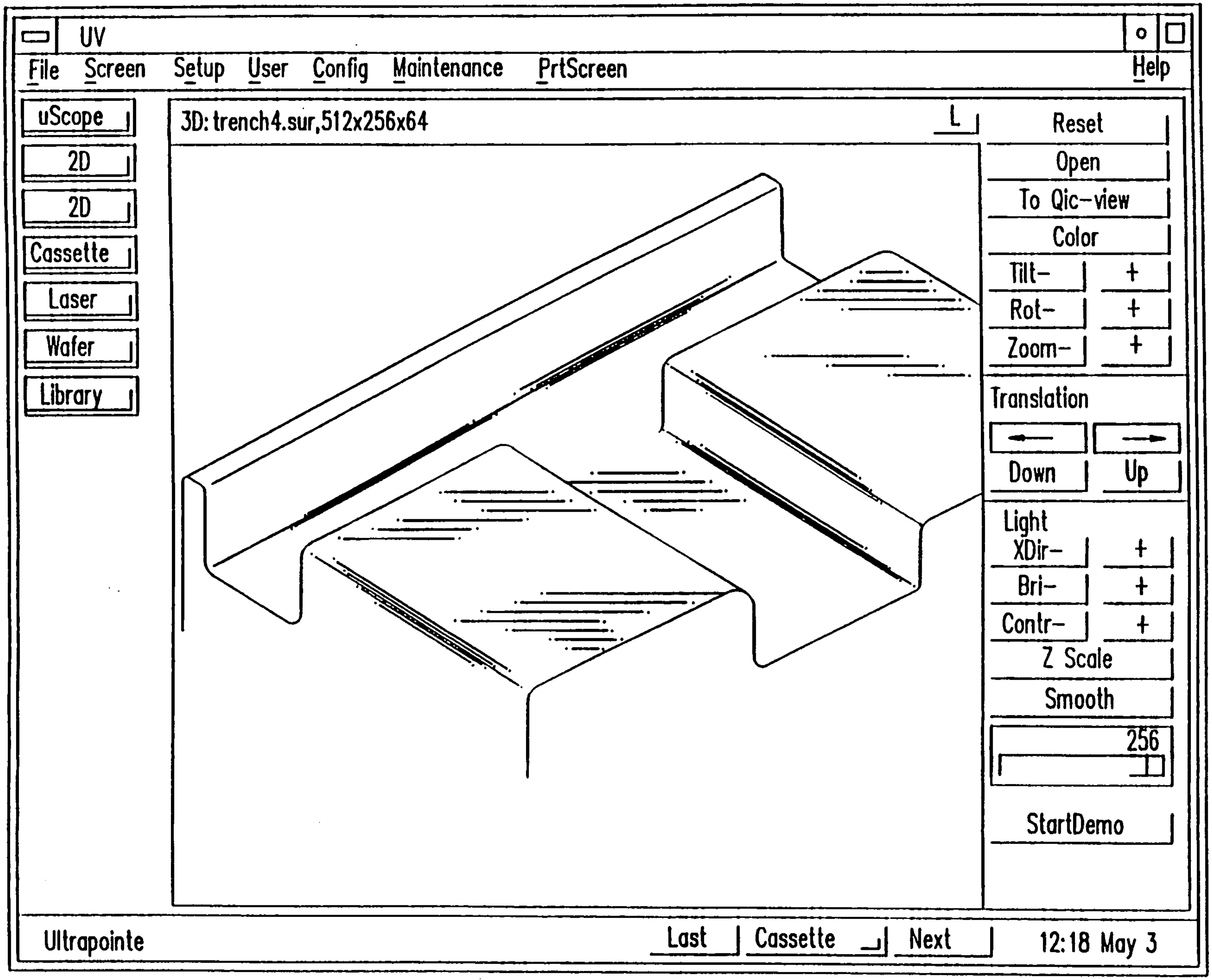

UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
uScope
2D
2D
Cassette
Laser
Wafer
Library
3D: trench4.sur,512x256x64
L
Reset
Open
To Qic-view
Color
Tilt-
+
Rot-
+
Zoom-
+
Translation
Down
Up
Light
XDir-
+
Bri-
+
Contr-
+
Z Scale
Smooth
256
StartDemo
Ultrapointe
Last
Cassette
Next
12:18 May 3

FIG. 42

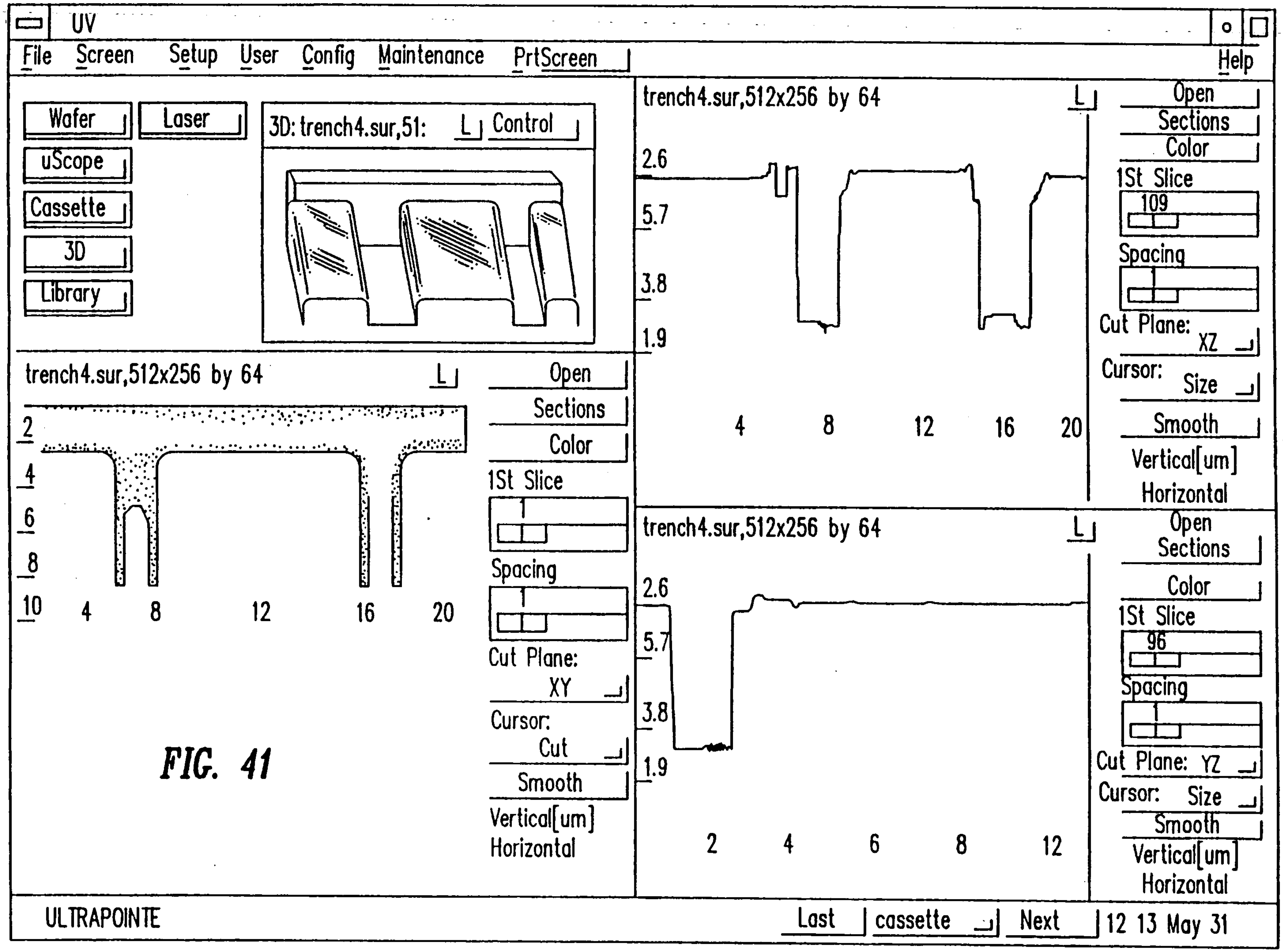
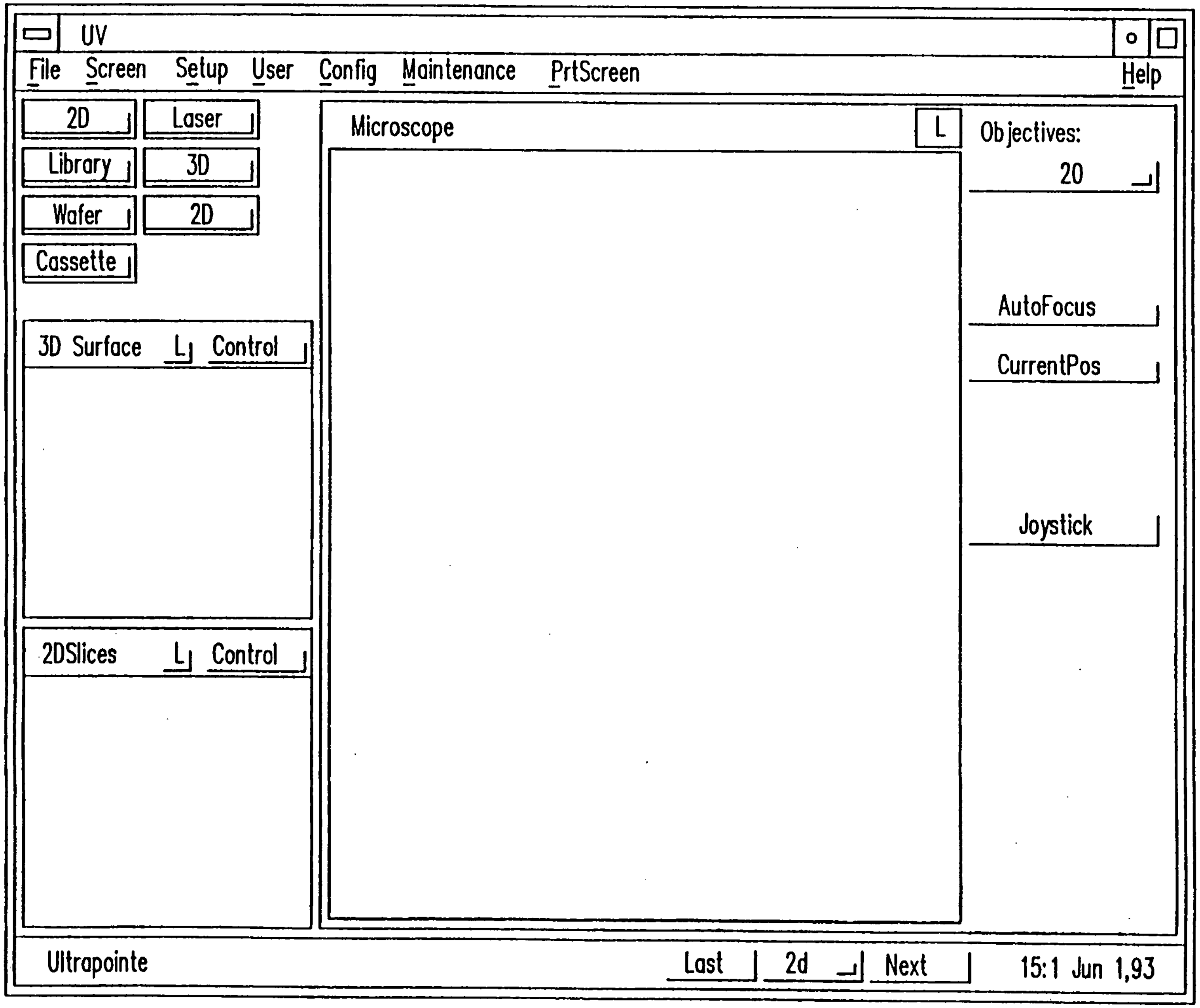
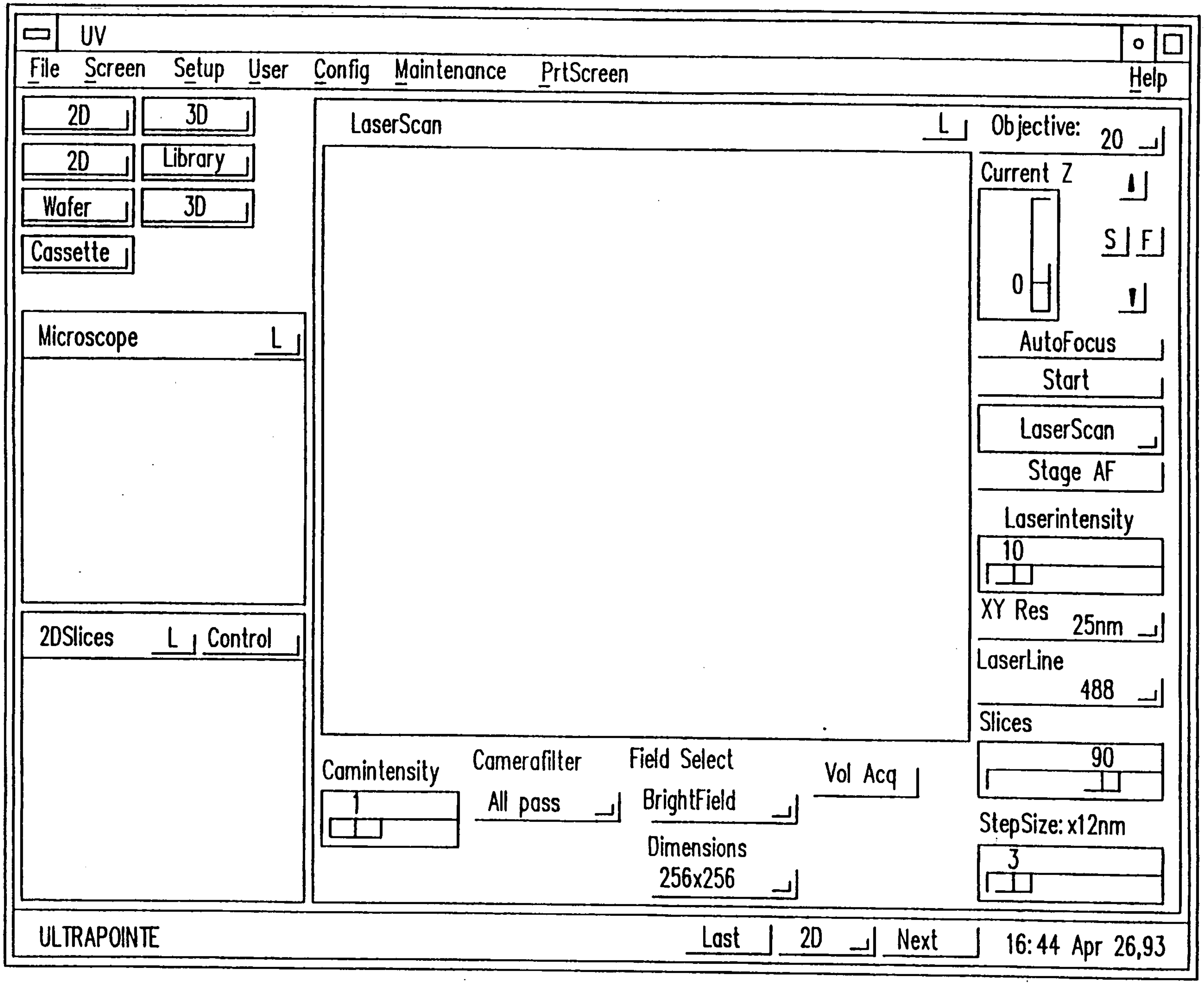

UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
Wafer
Laser
uScope
Cassette
3D
Library
3D: trench4.sur,51:
Control
trench4.sur,512x256 by 64
2.6
5.7
3.8
1.9
4
8
12
16
20
Open
Sections
Color
1St Slice
109
Spacing
Cut Plane:
XZ
Cursor:
Size
Smooth
Vertical[um]
Horizontal
trench4.sur,512x256 by 64
2
4
6
8
10
4
8
12
16
20
Open
Sections
Color
1St Slice
1
Spacing
1
Cut Plane:
XY
Cursor:
Cut
Smooth
Vertical[um]
Horizontal
trench4.sur,512x256 by 64
2.6
5.7
3.8
1.9
2
4
6
8
12
Open
Sections
Color
1St Slice
96
Spacing
1
Cut Plane: YZ
Cursor: Size
Smooth
Vertical[um]
Horizontal
ULTRAPOINTE
Last
cassette
Next
12 13 May 31

FIG. 41


UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
2D
Laser
Library
3D
Wafer
2D
Cassette
3D Surface
L
Control
2DSlices
L
Control
Microscope
L
Objectives:
20
AutoFocus
CurrentPos
Joystick
Ultrapointe
Last
2d
Next
15:1 Jun 1,93

Microscope
Window

FIG. 40


UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
2D
3D
2D
Library
Wafer
3D
Cassette
Microscope
L
2DSlices
L
Control
LaserScan
L
Current Z
0
S
F
AutoFocus
Start
LaserScan
Stage AF
Vol Acq
ULTRAPOINTE
Last
2D
Next

Objective: 20

Laserintensity 10

XY Res 25nm

LaserLine 488

Slices 90

StepSize: x12nm 3

Camintensity 1

Camerafilter All pass

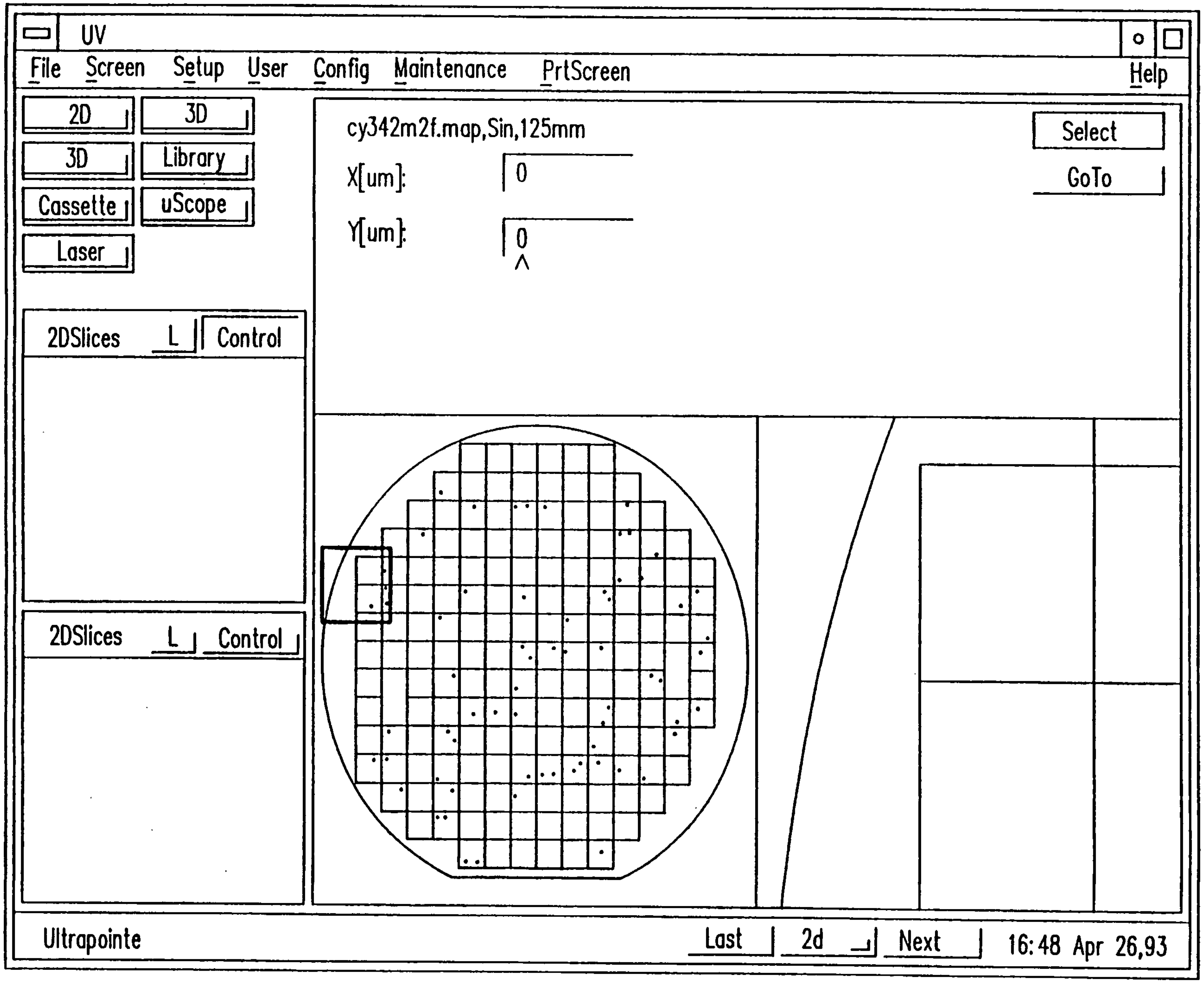
Field Select BrightField

Dimensions 256x256

16:44 Apr 26,93

Laser Scan Window

FIG. 39


UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
2D
3D
3D
Library
Cassette
uScope
Laser
2DSlices
L
Control
2DSlices
L
Control
Select
GoTo
Ultrapointe
Last
2d
Next

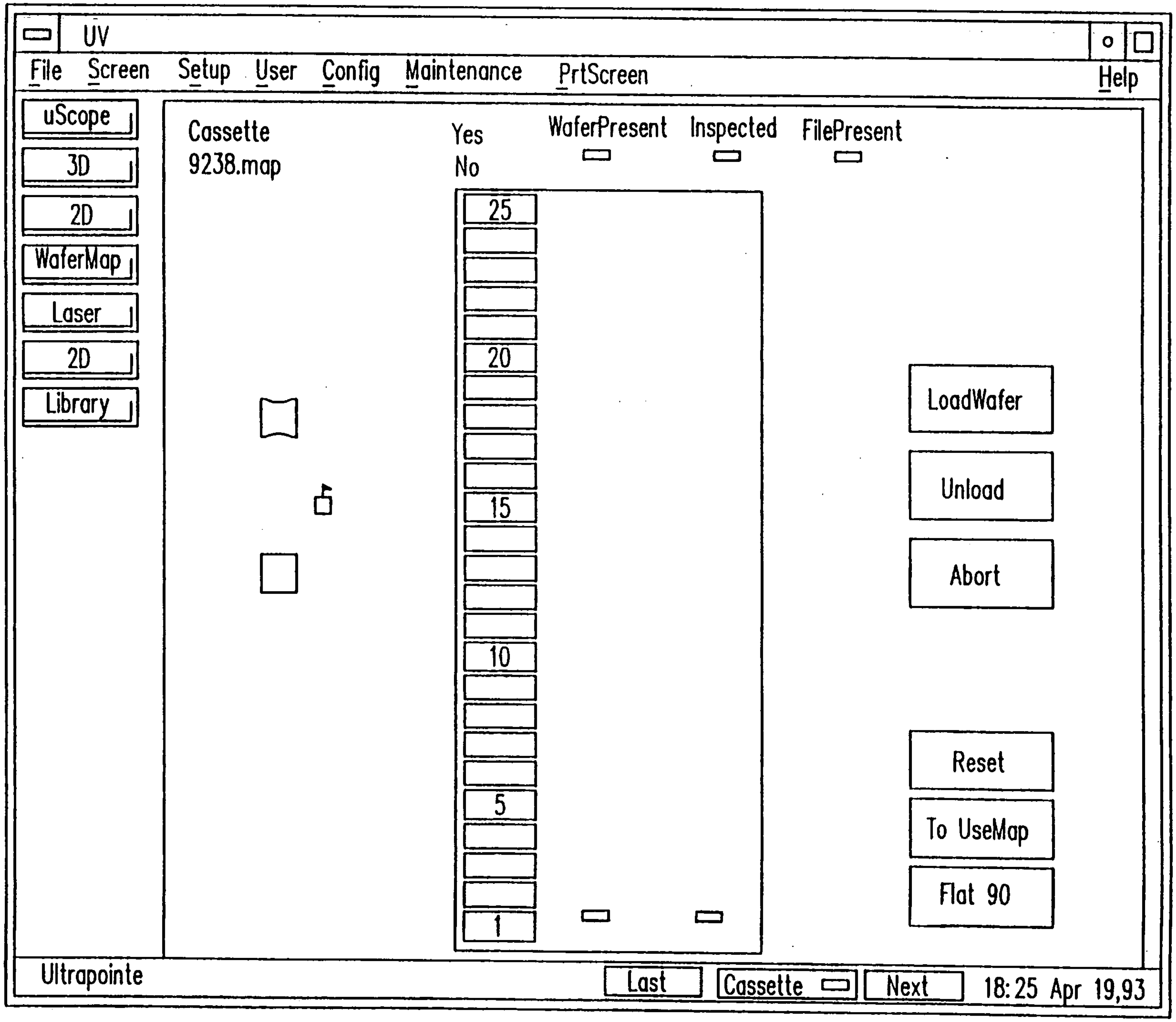
cy342m2f.map,Sin,125mm

X[um]: 0

Y[um]: 0

16:48 Apr 26,93

FIG. 38



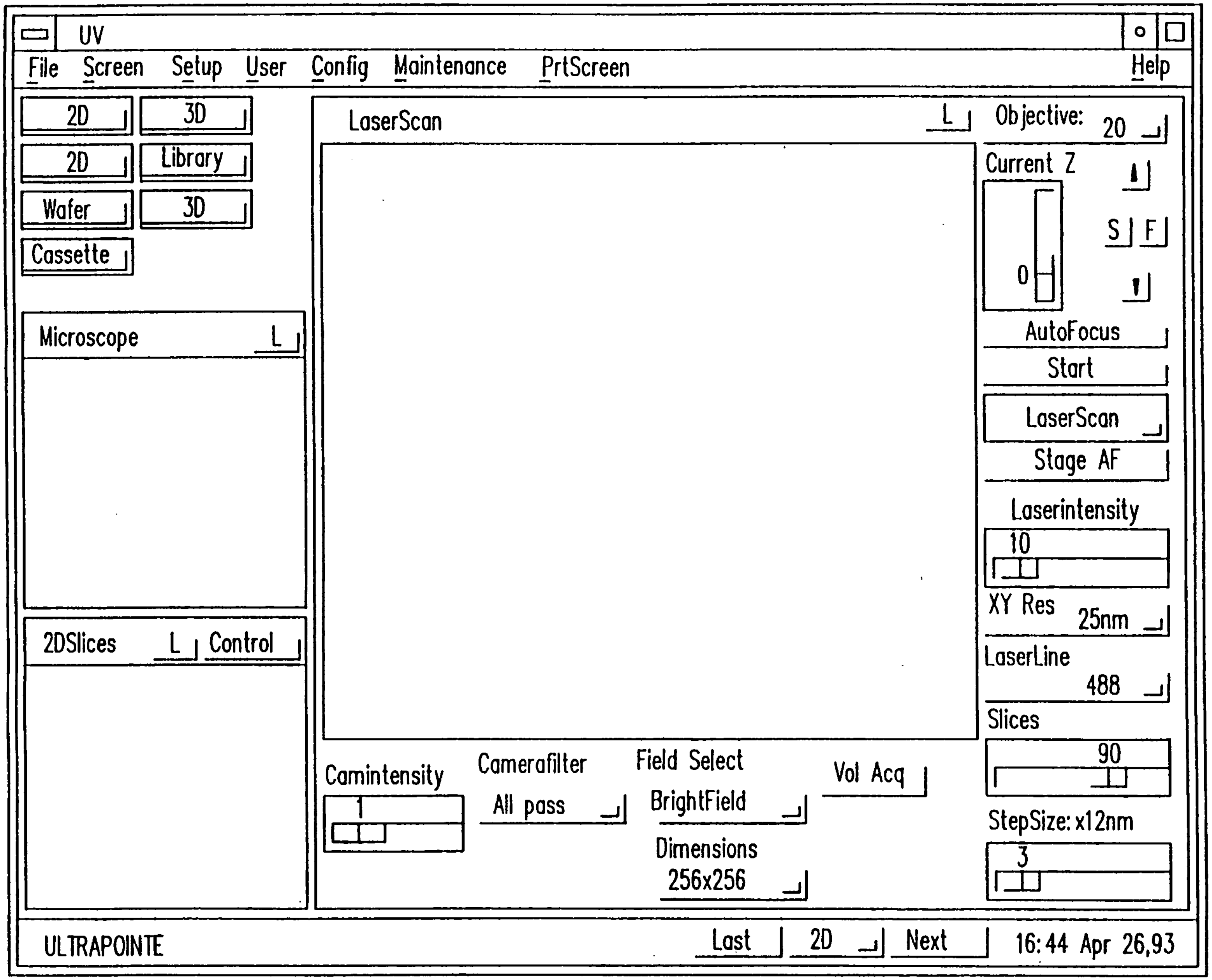
UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
uScope
3D
2D
WaferMap
Laser
2D
Library
Yes
No
WaferPresent
Inspected
FilePresent
25
20
15
10
5
1
LoadWafer
Unload
Abort
Reset
To UseMap
Flat 90
Ultrapointe
Last
Cassette
Next

Cassette
9238.map

18:25 Apr 19,93

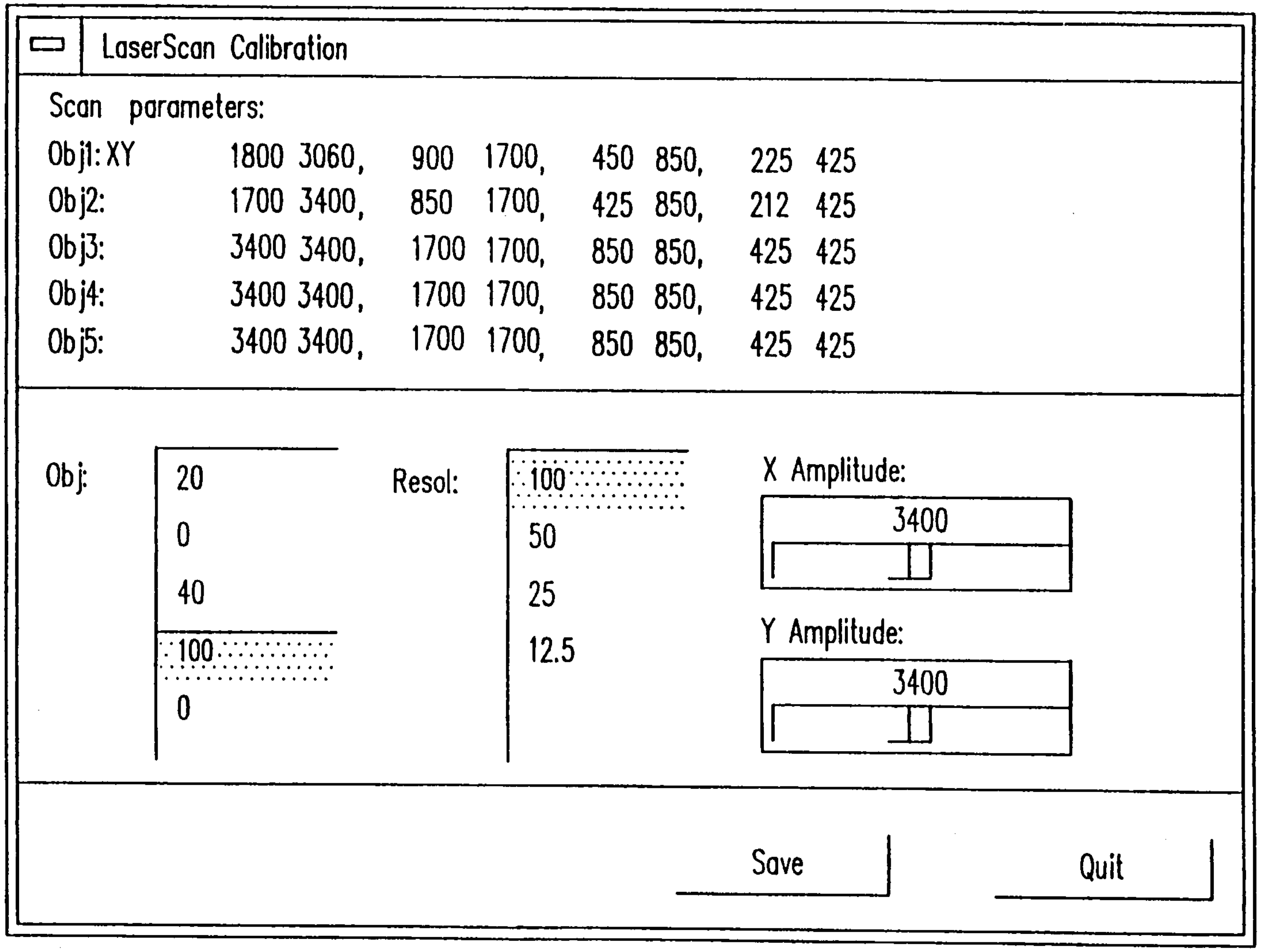
Cassette
Window

FIG. 37


UV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
2D
3D
2D
Library
Wafer
3D
Cassette
Microscope
L
2DSlices
L
Control
LaserScan
L
Current Z
0
S
F
AutoFocus
Start
LaserScan
Stage AF
Vol Acq
Last
2D
Next
ULTRAPOINTE

Objective: 20
Laserintensity 10
XY Res 25nm
LaserLine 488
Slices 90
StepSize: x12nm 3
Camintensity 1
Camerafilter All pass
Field Select BrightField
Dimensions 256x256
16:44 Apr 26,93

FIG. 36


LaserScan Calibration

```
Scan  parameters:
Obj1: XY      1800 3060,   900  1700,   450  850,   225  425
Obj2:         1700 3400,   850  1700,   425  850,   212  425
Obj3:         3400 3400,   1700 1700,   850  850,   425  425
Obj4:         3400 3400,   1700 1700,   850  850,   425  425
Obj5:         3400 3400,   1700 1700,   850  850,   425  425
```

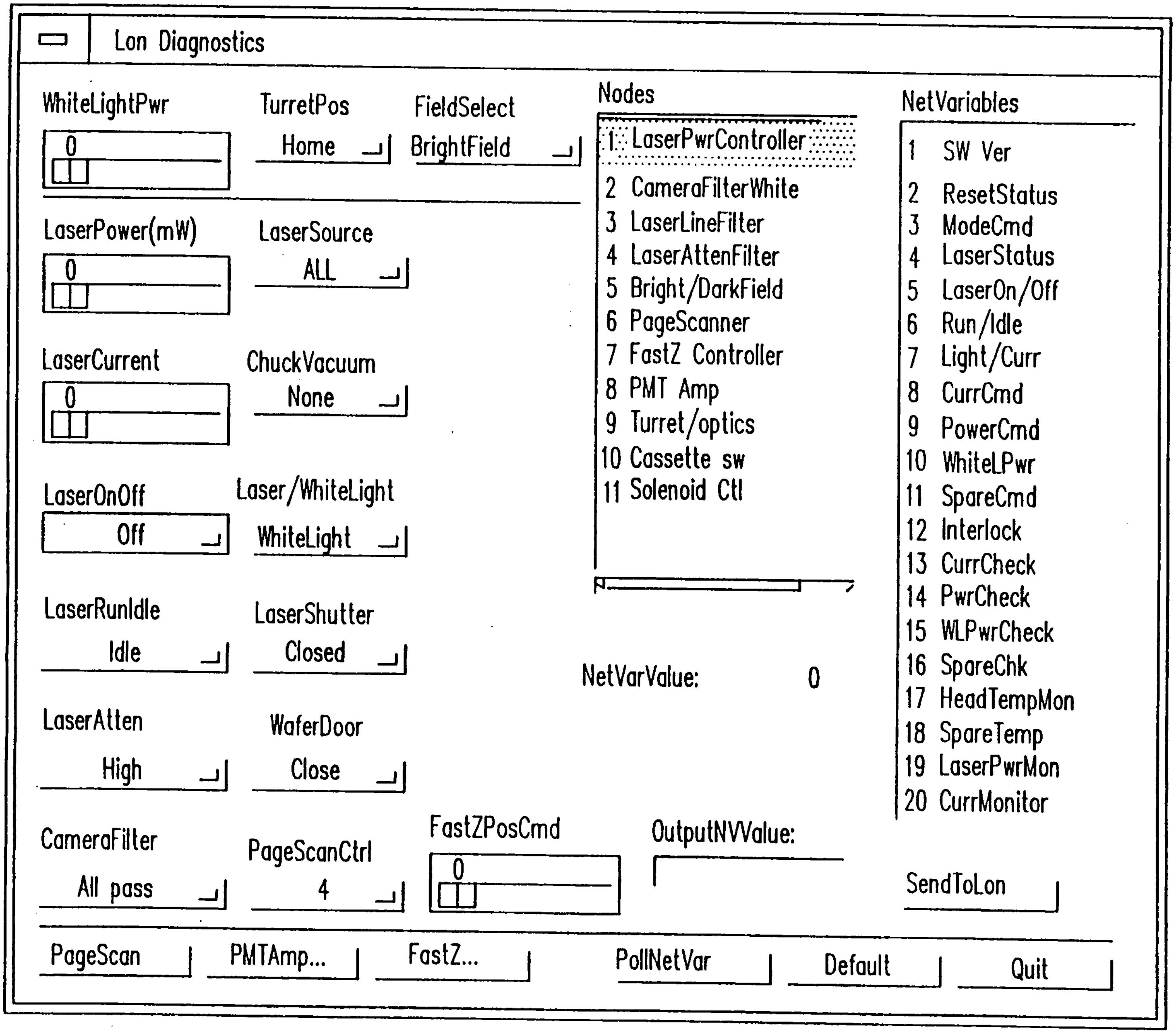
Obj: 20, 0, 40, 100, 0

Resol: 100, 50, 25, 12.5

X Amplitude: 3400

Y Amplitude: 3400

FIG. 35



Lon Diagnostics

WhiteLightPwr: 0

TurretPos: Home

FieldSelect: BrightField

LaserPower(mW): 0

LaserSource: ALL

LaserCurrent: 0

ChuckVacuum: None

LaserOnOff: Off

Laser/WhiteLight: WhiteLight

LaserRunIdle: Idle

LaserShutter: Closed

LaserAtten: High

WaferDoor: Close

CameraFilter: All pass

PageScanCtrl: 4

FastZPosCmd: 0

Nodes

1 LaserPwrController
2 CameraFilterWhite
3 LaserLineFilter
4 LaserAttenFilter
5 Bright/DarkField
6 PageScanner
7 FastZ Controller
8 PMT Amp
9 Turret/optics
10 Cassette sw
11 Solenoid Ctl

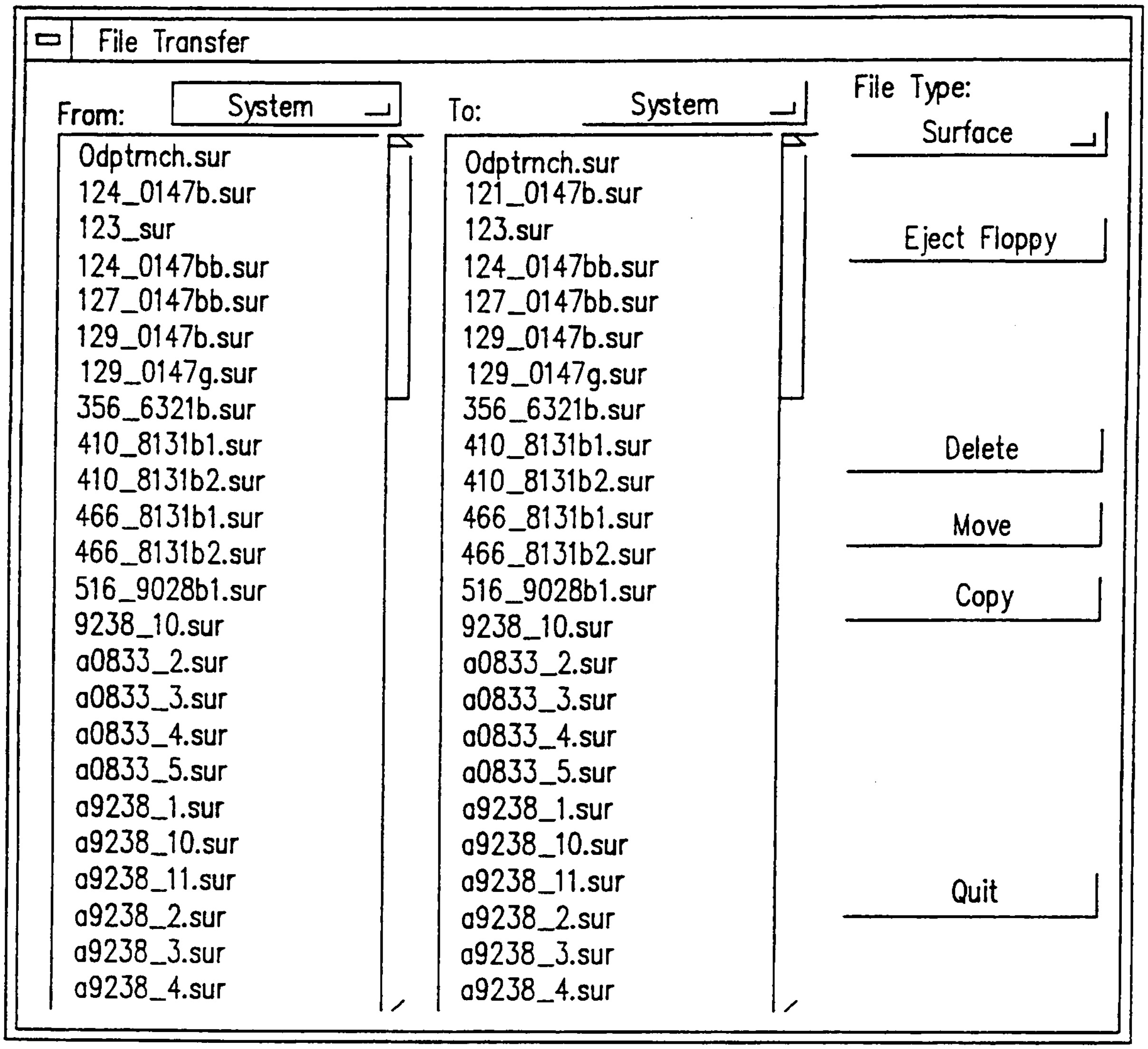
NetVariables

1 SW Ver
2 ResetStatus
3 ModeCmd
4 LaserStatus
5 LaserOn/Off
6 Run/Idle
7 Light/Curr
8 CurrCmd
9 PowerCmd
10 WhiteLPwr
11 SpareCmd
12 Interlock
13 CurrCheck
14 PwrCheck
15 WLPwrCheck
16 SpareChk
17 HeadTempMon
18 SpareTemp
19 LaserPwrMon
20 CurrMonitor

NetVarValue: 0

OutputNVValue:

FIG. 34


File Transfer
Eject Floppy
Delete
Move
Copy
Quit

From: System

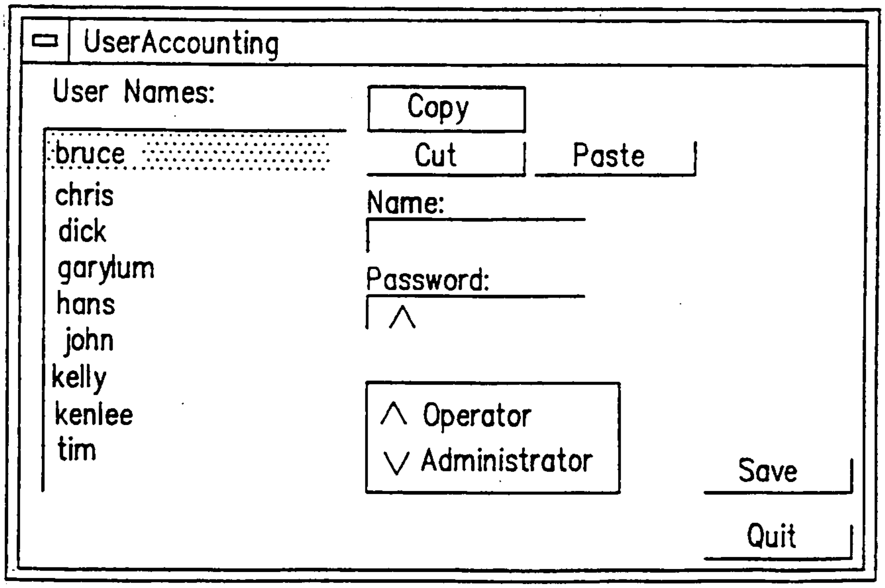
Odptrnch.sur
124_0147b.sur
123_sur
124_0147bb.sur
127_0147bb.sur
129_0147b.sur
129_0147g.sur
356_6321b.sur
410_8131b1.sur
410_8131b2.sur
466_8131b1.sur
466_8131b2.sur
516_9028b1.sur
9238_10.sur
a0833_2.sur
a0833_3.sur
a0833_4.sur
a0833_5.sur
a9238_1.sur
a9238_10.sur
a9238_11.sur
a9238_2.sur
a9238_3.sur
a9238_4.sur

To: System

Odptrnch.sur
121_0147b.sur
123.sur
124_0147bb.sur
127_0147bb.sur
129_0147b.sur
129_0147g.sur
356_6321b.sur
410_8131b1.sur
410_8131b2.sur
466_8131b1.sur
466_8131b2.sur
516_9028b1.sur
9238_10.sur
a0833_2.sur
a0833_3.sur
a0833_4.sur
a0833_5.sur
a9238_1.sur
a9238_10.sur
a9238_11.sur
a9238_2.sur
a9238_3.sur
a9238_4.sur

File Type: Surface

FIG. 32


UserAccounting
Copy
Cut
Paste
Save
Quit

User Names:

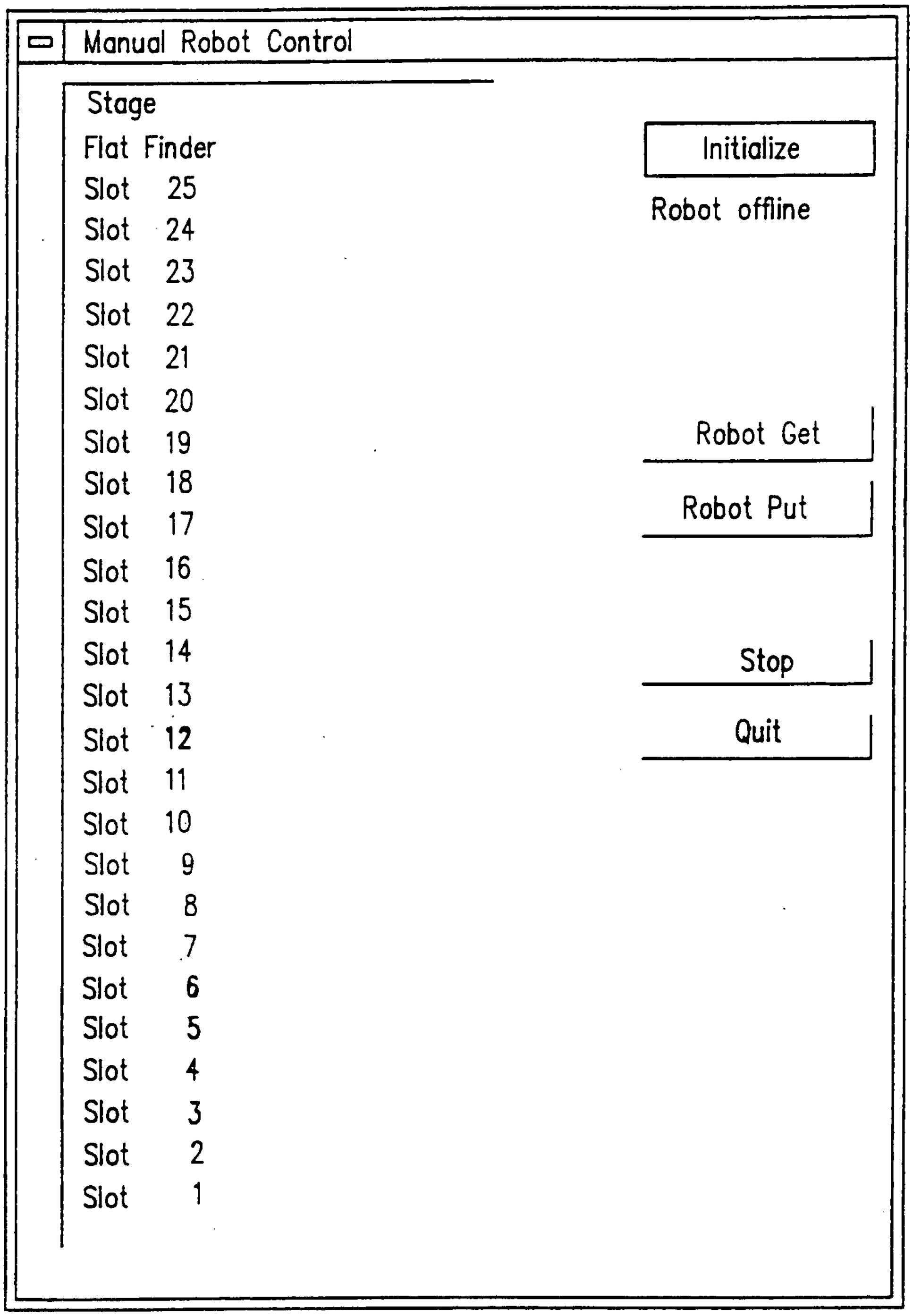
bruce
chris
dick
garylum
hans
john
kelly
kenlee
tim

Name:

Password:

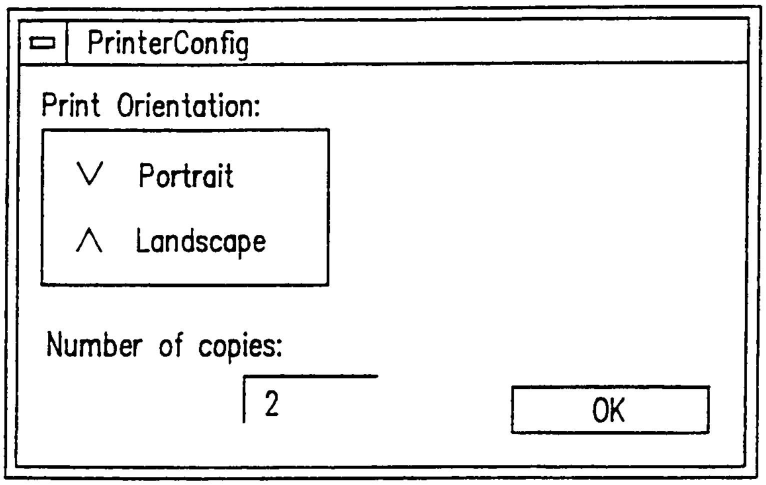
∧ Operator
∨ Administrator

FIG. 33


Manual Robot Control
Stage
Flat Finder
Slot 25
Slot 24
Slot 23
Slot 22
Slot 21
Slot 20
Slot 19
Slot 18
Slot 17
Slot 16
Slot 15
Slot 14
Slot 13
Slot 12
Slot 11
Slot 10
Slot 9
Slot 8
Slot 7
Slot 6
Slot 5
Slot 4
Slot 3
Slot 2
Slot 1
Initialize
Robot Get
Robot Put
Stop
Quit

Robot offline

FIG. 31



PrinterConfig

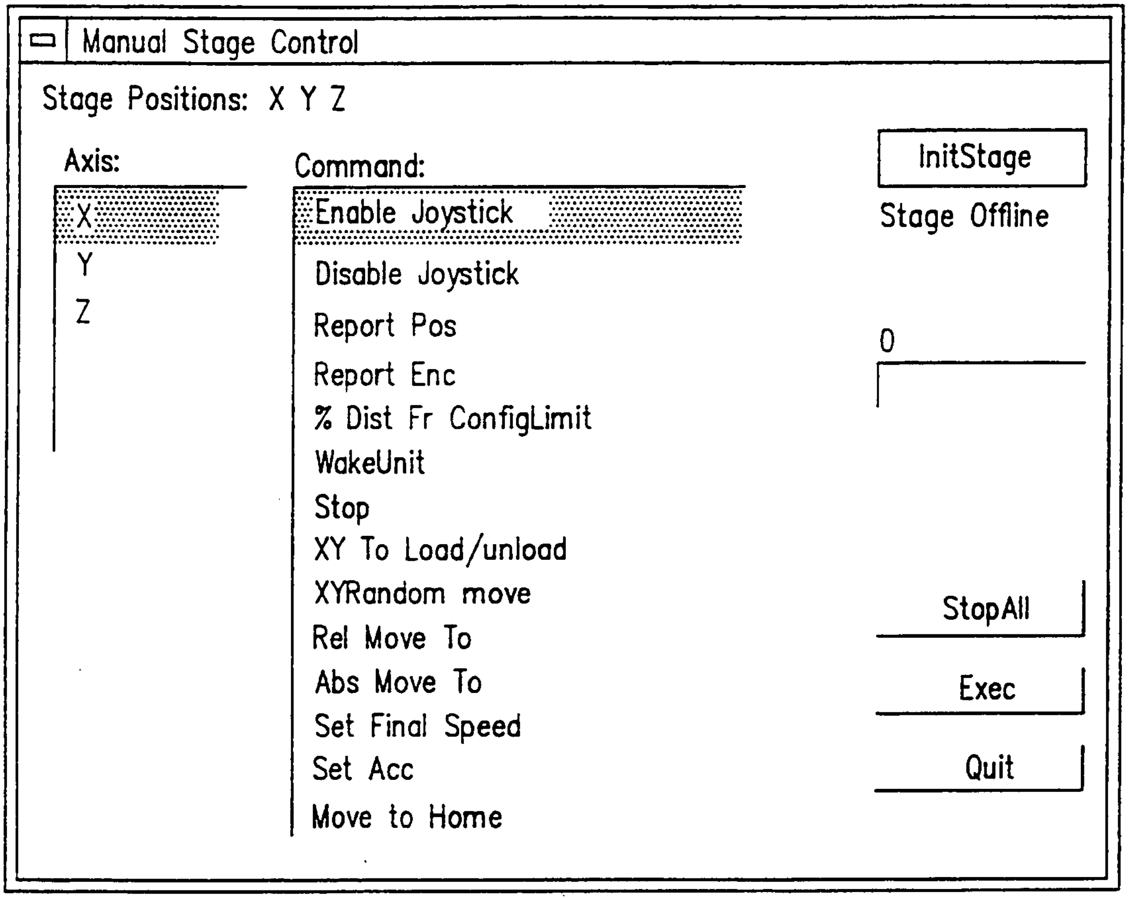
Print Orientation:

V Portrait

Λ Landscape

Number of copies: 2

FIG. 29



Manual Stage Control

Stage Positions: X Y Z

Axis:

X
Y
Z

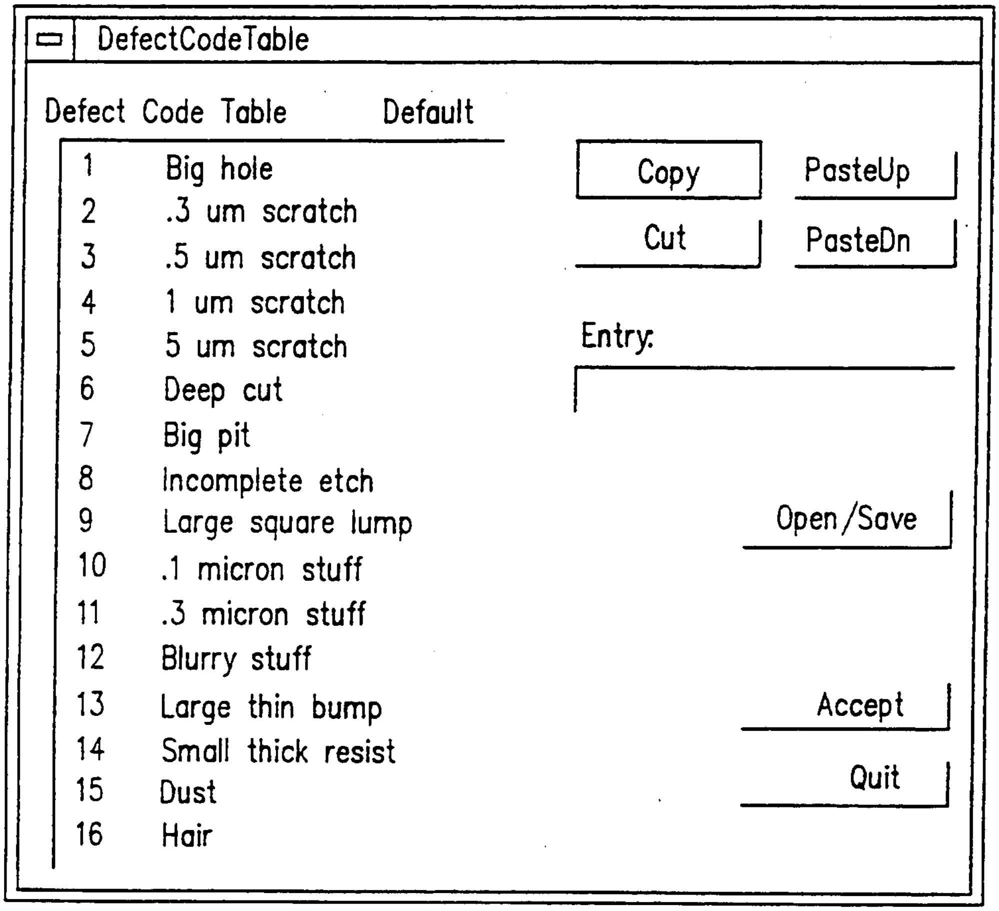
Command:

Enable Joystick
Disable Joystick
Report Pos
Report Enc
% Dist Fr ConfigLimit
WakeUnit
Stop
XY To Load/unload
XYRandom move
Rel Move To
Abs Move To
Set Final Speed
Set Acc
Move to Home
InitStage
StopAll
Exec
Quit

Stage Offline

0

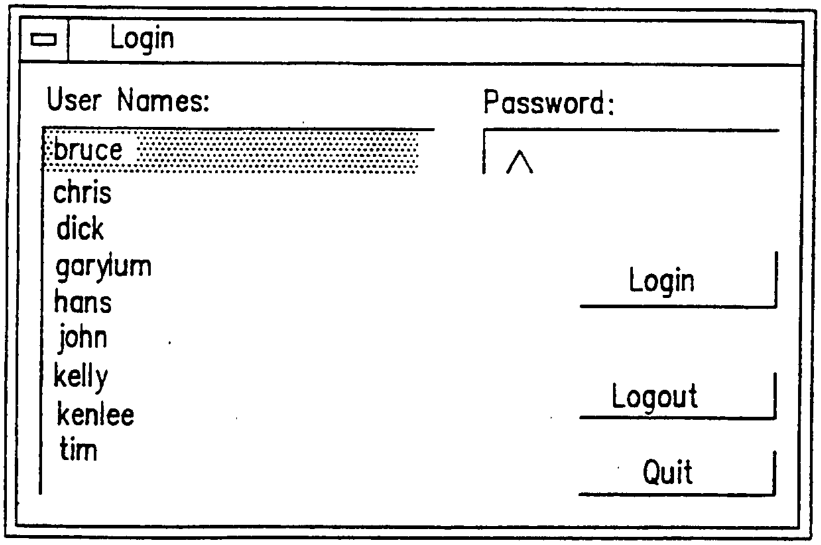
FIG. 30


DefectCodeTable
Copy
PasteUp
Cut
PasteDn
Entry:
Open/Save
Accept
Quit

Defect Code Table Default

1	Big hole
2	.3 um scratch
3	.5 um scratch
4	1 um scratch
5	5 um scratch
6	Deep cut
7	Big pit
8	Incomplete etch
9	Large square lump
10	.1 micron stuff
11	.3 micron stuff
12	Blurry stuff
13	Large thin bump
14	Small thick resist
15	Dust
16	Hair

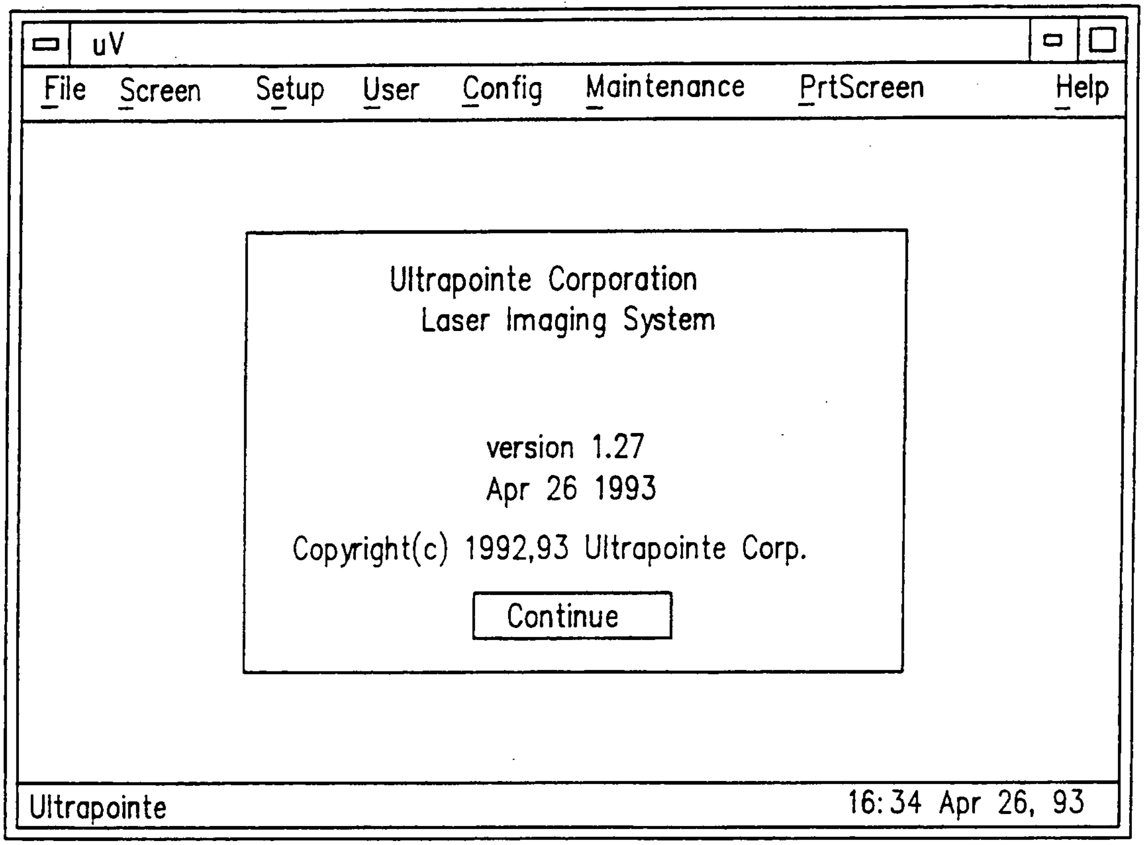
FIG. 27


Login
Password:
^
Login
Logout
Quit

User Names:

- bruce
- chris
- dick
- garylum
- hans
- john
- kelly
- kenlee
- tim

FIG. 28


uV
File
Screen
Setup
User
Config
Maintenance
PrtScreen
Help
Continue

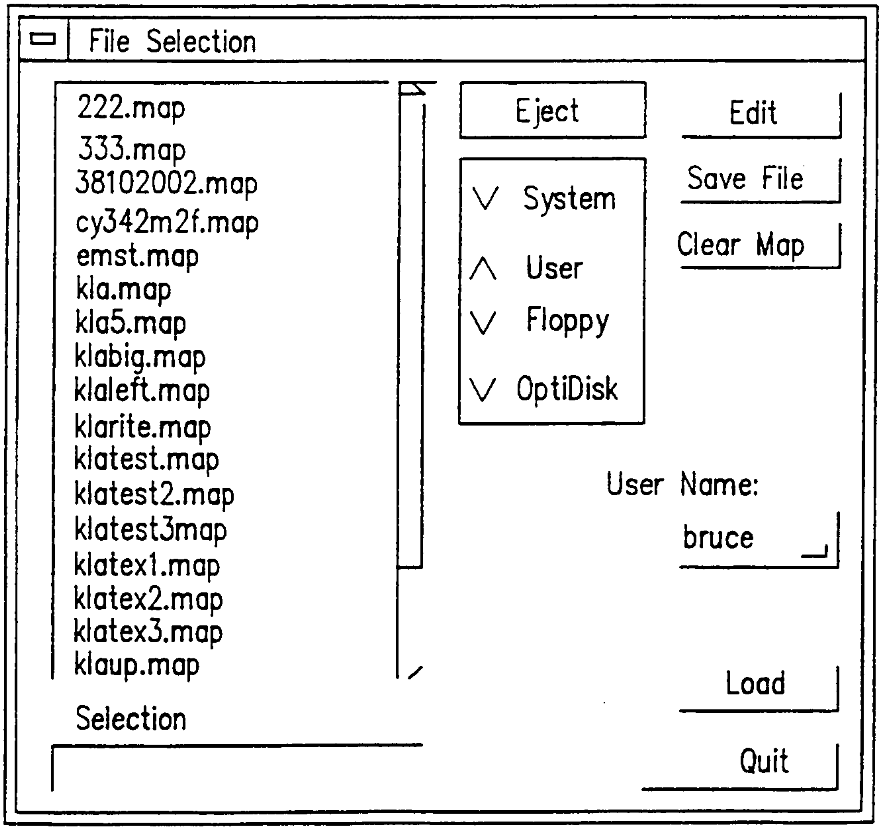
Ultrapointe Corporation
Laser Imaging System

version 1.27
Apr 26 1993

Copyright(c) 1992,93 Ultrapointe Corp.

Ultrapointe 16:34 Apr 26, 93

FIG. 25


File Selection
Eject
System
User
Floppy
OptiDisk
Edit
Save File
Clear Map
Load
Quit
Selection

222.map
333.map
38102002.map
cy342m2f.map
emst.map
kla.map
kla5.map
klabig.map
klaleft.map
klarite.map
klatest.map
klatest2.map
klatest3map
klatex1.map
klatex2.map
klatex3.map
klaup.map

User Name: bruce

FIG. 26

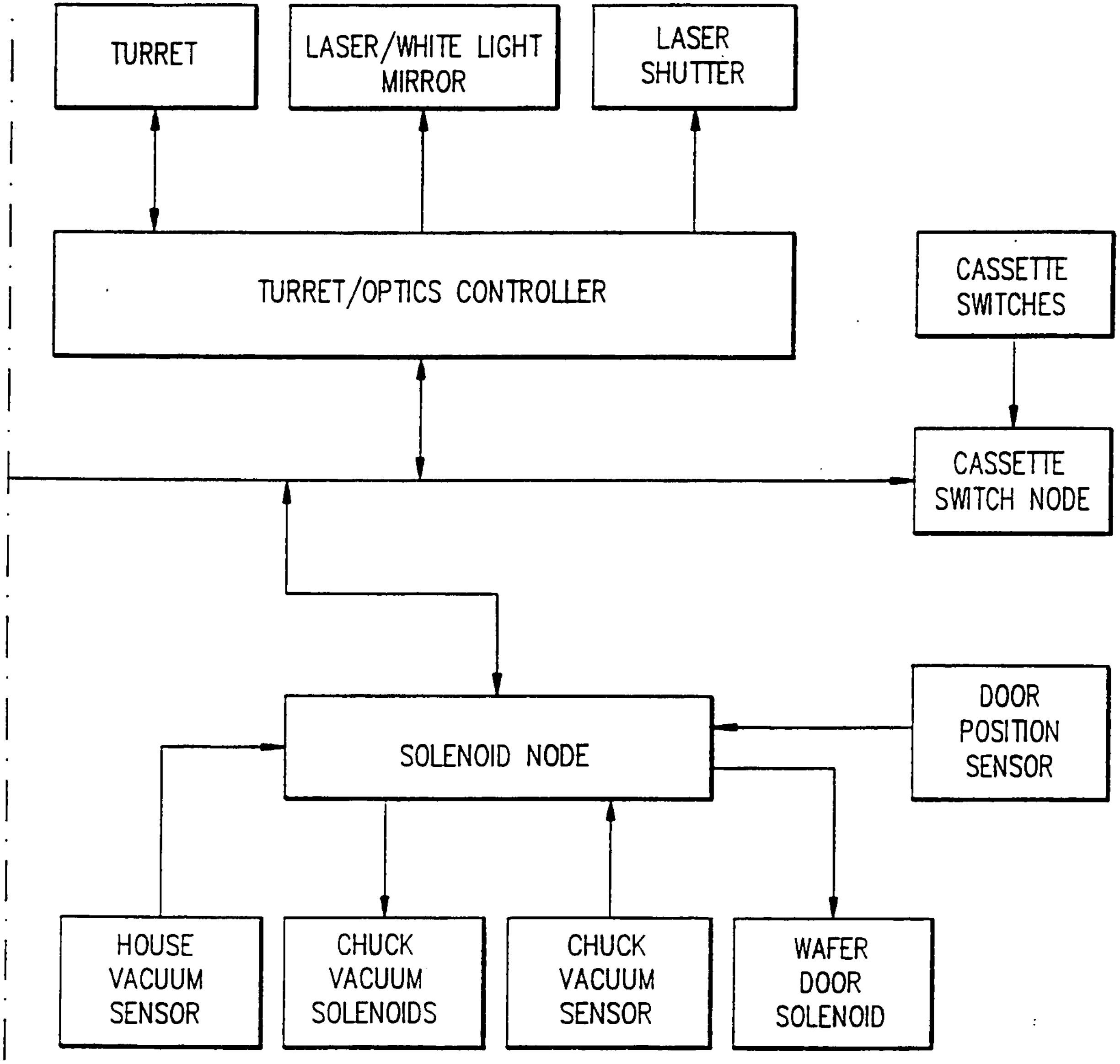

TURRET
LASER/WHITE LIGHT MIRROR
LASER SHUTTER
TURRET/OPTICS CONTROLLER
CASSETTE SWITCHES
CASSETTE SWITCH NODE
DOOR POSITION SENSOR
SOLENOID NODE
HOUSE VACUUM SENSOR
CHUCK VACUUM SOLENOIDS
CHUCK VACUUM SENSOR
WAFER DOOR SOLENOID

FIG. 24-3

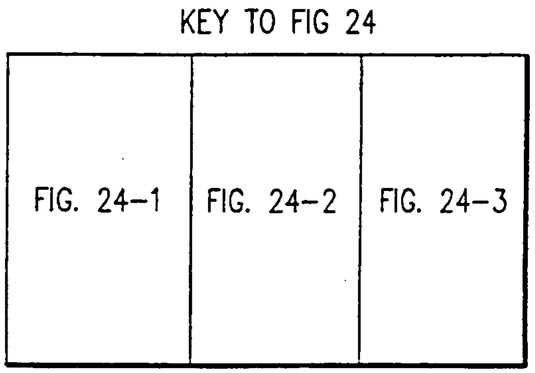

KEY TO FIG 24
FIG. 24-1
FIG. 24-2
FIG. 24-3

FIG. 24

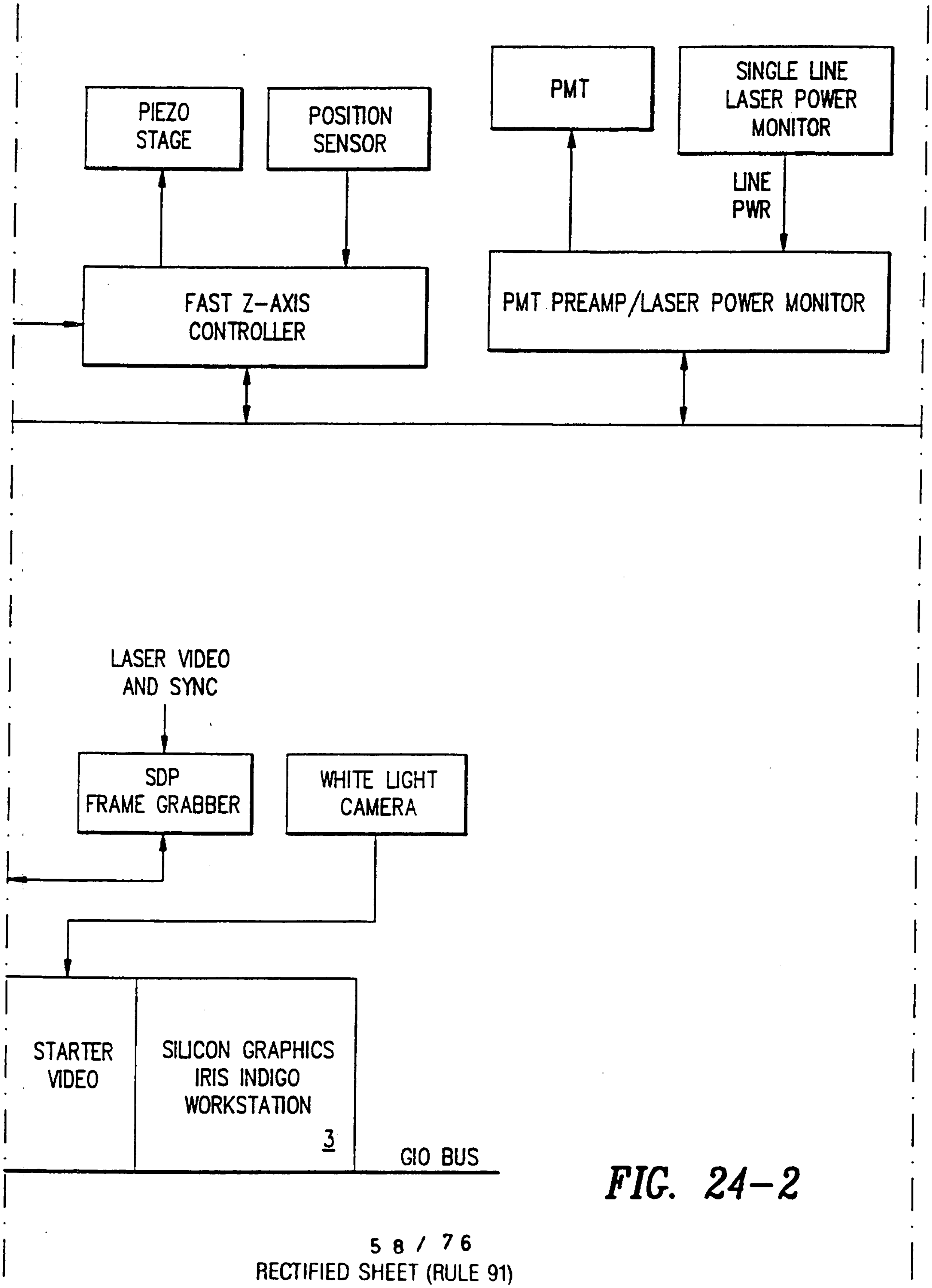

PIEZO STAGE
POSITION SENSOR
PMT
SINGLE LINE LASER POWER MONITOR
LINE PWR
FAST Z-AXIS CONTROLLER
PMT PREAMP/LASER POWER MONITOR
LASER VIDEO AND SYNC
SDP FRAME GRABBER
WHITE LIGHT CAMERA
STARTER VIDEO
SILICON GRAPHICS IRIS INDIGO WORKSTATION
3
GIO BUS

FIG. 24-2

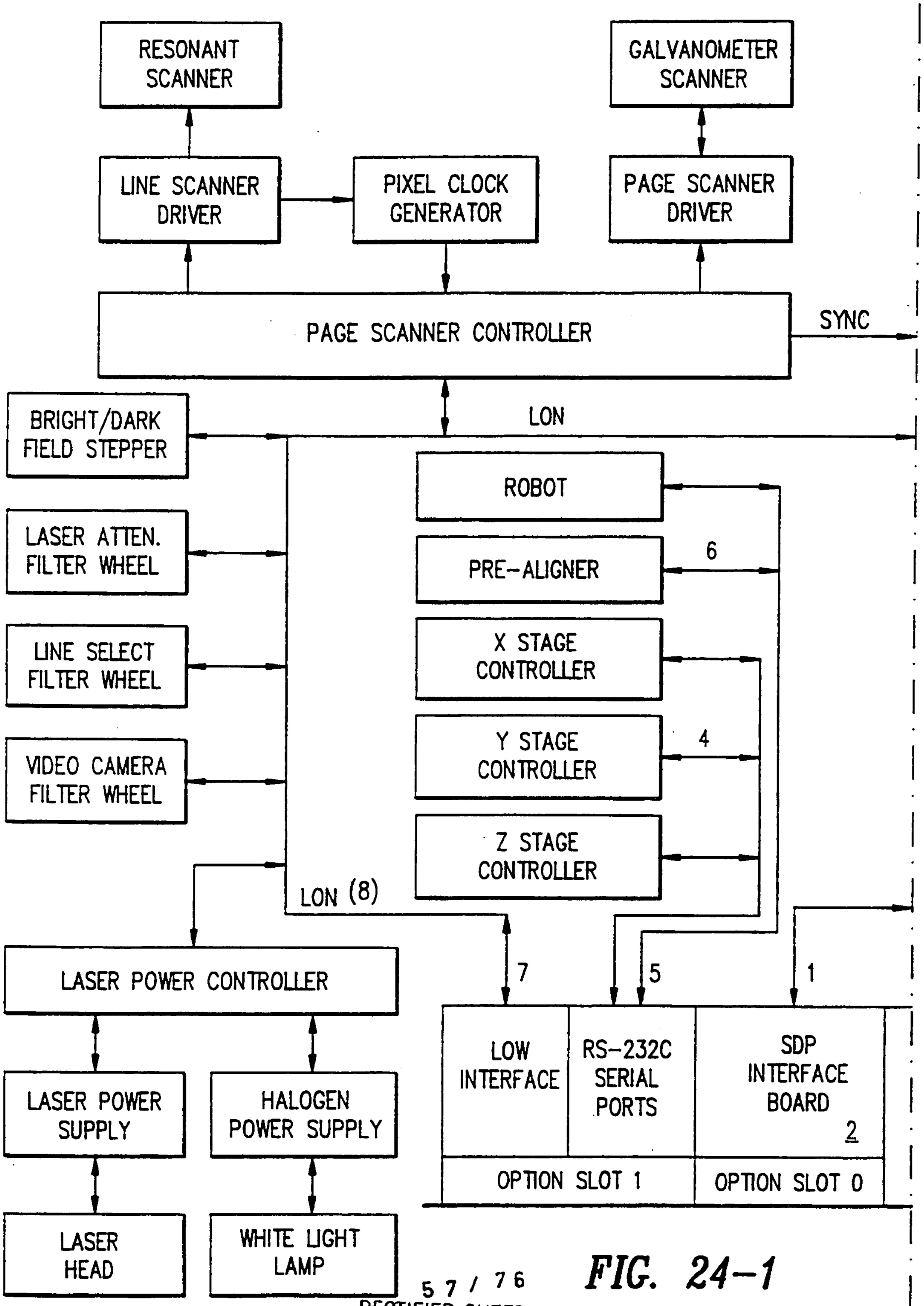

RESONANT SCANNER
GALVANOMETER SCANNER
LINE SCANNER DRIVER
PIXEL CLOCK GENERATOR
PAGE SCANNER DRIVER
PAGE SCANNER CONTROLLER
SYNC
LON
BRIGHT/DARK FIELD STEPPER
LASER ATTEN. FILTER WHEEL
LINE SELECT FILTER WHEEL
VIDEO CAMERA FILTER WHEEL
ROBOT
PRE-ALIGNER
6
X STAGE CONTROLLER
Y STAGE CONTROLLER
4
Z STAGE CONTROLLER
LON (8)
7
5
1
LASER POWER CONTROLLER
LOW INTERFACE
RS-232C SERIAL PORTS
SDP INTERFACE BOARD
2
OPTION SLOT 1
OPTION SLOT 0
LASER POWER SUPPLY
HALOGEN POWER SUPPLY
LASER HEAD
WHITE LIGHT LAMP

FIG. 24-1

RECTIFIED SHEET (RULE 91)

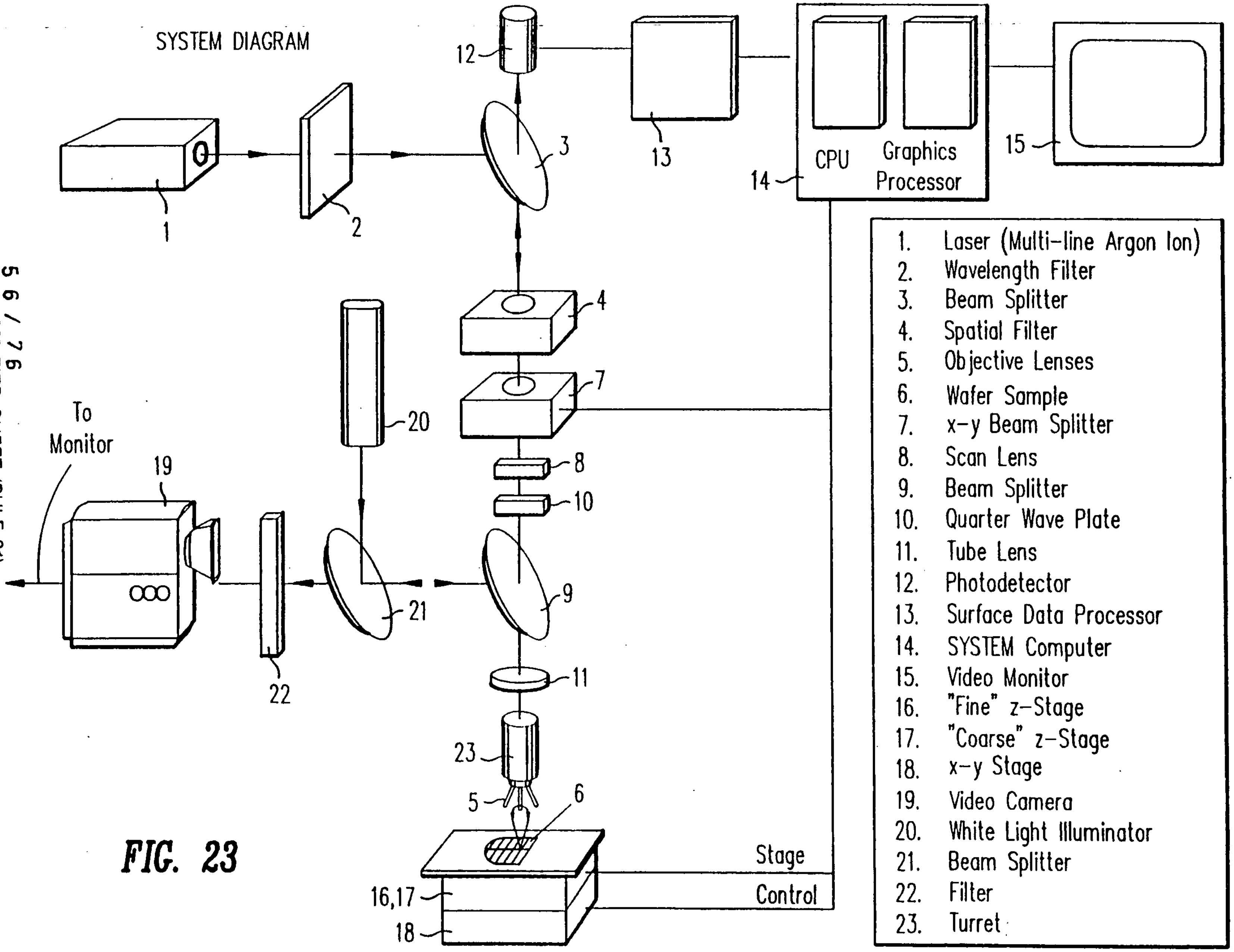

SYSTEM DIAGRAM
1
2
3
12
13
14
CPU
Graphics Processor
15
4
7
8
10
9
11
23
5
6
16,17
18
20
21
22
19
To Monitor
Stage Control
1. Laser (Multi-line Argon Ion)
2. Wavelength Filter
3. Beam Splitter
4. Spatial Filter
5. Objective Lenses
6. Wafer Sample
7. x-y Beam Splitter
8. Scan Lens
9. Beam Splitter
10. Quarter Wave Plate
11. Tube Lens
12. Photodetector
13. Surface Data Processor
14. SYSTEM Computer
15. Video Monitor
16. "Fine" z-Stage
17. "Coarse" z-Stage
18. x-y Stage
19. Video Camera
20. White Light Illuminator
21. Beam Splitter
22. Filter
23. Turret

FIG. 23

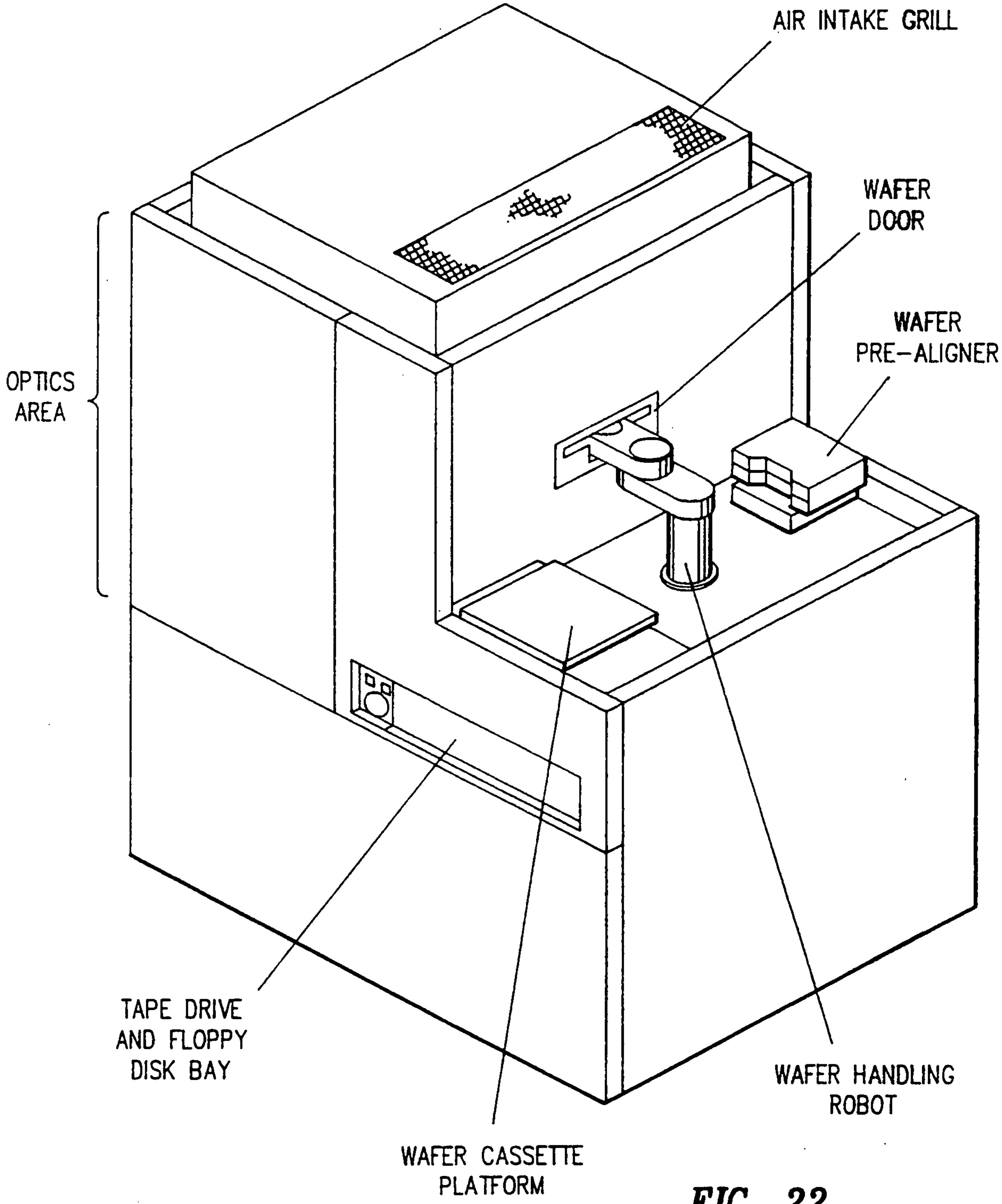

AIR INTAKE GRILL
WAFER DOOR
WAFER PRE-ALIGNER
OPTICS AREA
TAPE DRIVE AND FLOPPY DISK BAY
WAFER HANDLING ROBOT
WAFER CASSETTE PLATFORM

FIG. 22

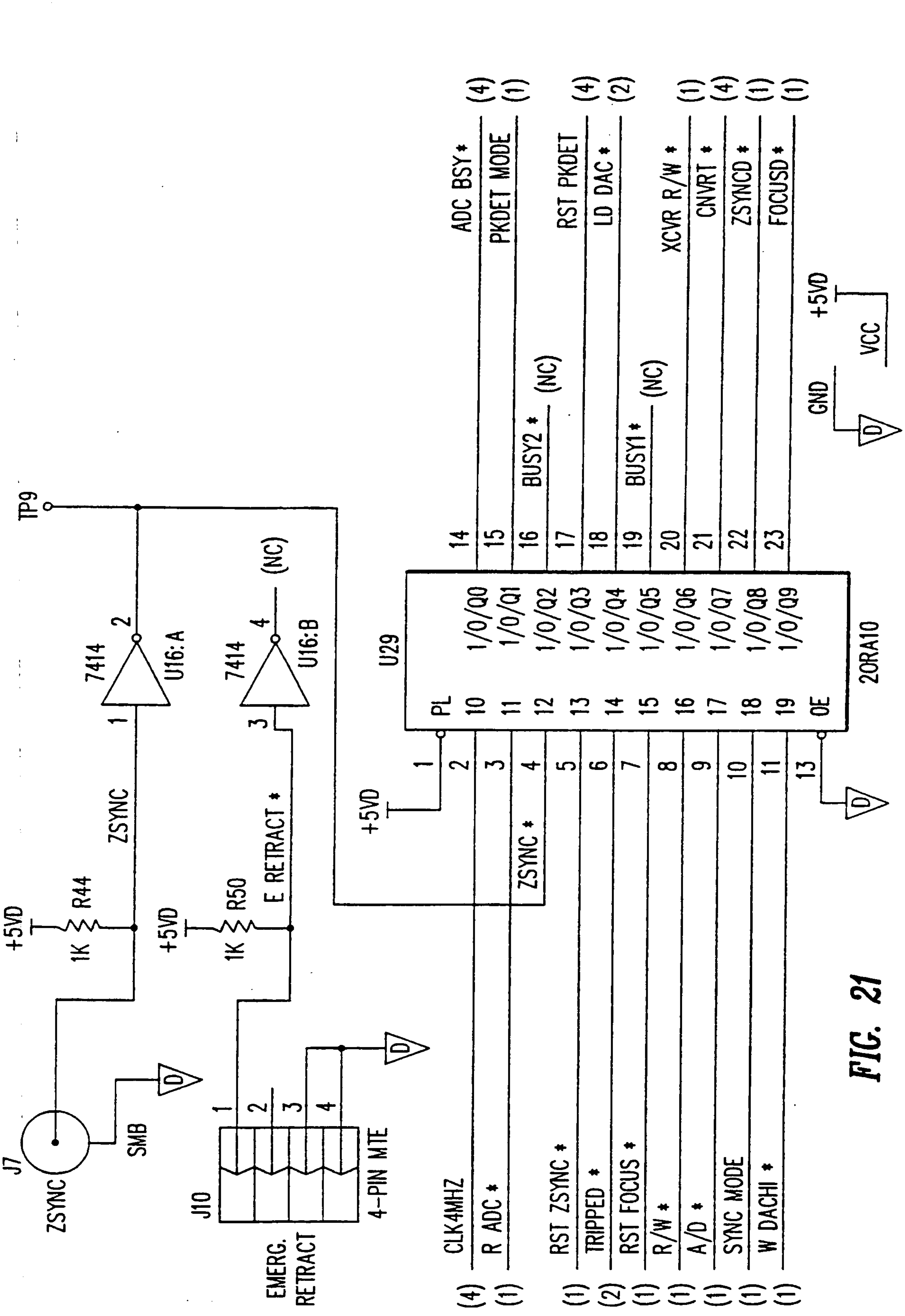

TP9
ZSYNC
J7
SMB
D
+5VD
1K
R44
ZSYNC
1
7414
2
U16:A
J10
4-PIN MTE
1
2
3
4
EMERG. RETRACT
+5VD
1K
R50
E RETRACT *
3
7414
4
U16:B
(NC)
+5VD
U29
20RA10
PL
1/0/Q0
1/0/Q1
1/0/Q2
1/0/Q3
1/0/Q4
1/0/Q5
1/0/Q6
1/0/Q7
1/0/Q8
1/0/Q9
OE
ZSYNC *
(4) CLK4MHZ
(1) R ADC *
(1) RST ZSYNC *
(2) TRIPPED *
(1) RST FOCUS *
(1) R/W *
(1) A/D *
(1) SYNC MODE
(1) W DACHI *
(4) ADC BSY *
(1) PKDET MODE
BUSY2 * (NC)
(4) RST PKDET
(2) LD DAC *
BUSY1 * (NC)
(1) XCVR R/W *
(4) CNVRT *
(1) ZSYNCD *
(1) FOCUSD *
GND
VCC
+5VD
D

FIG. 21

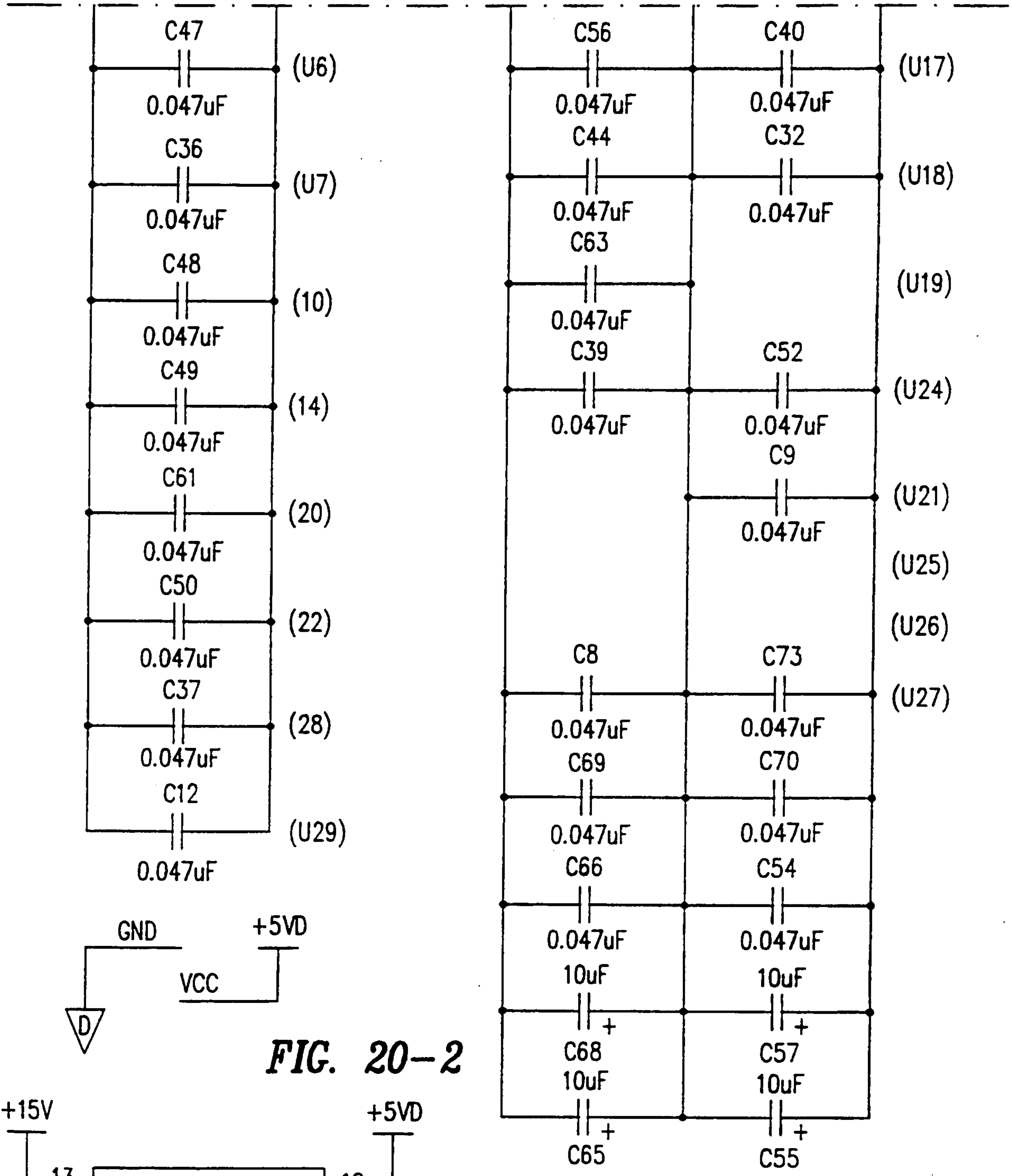

C47
0.047uF
(U6)
C36
0.047uF
(U7)
C48
0.047uF
(10)
C49
0.047uF
(14)
C61
0.047uF
(20)
C50
0.047uF
(22)
C37
0.047uF
(28)
C12
0.047uF
(U29)
GND
+5VD
VCC
D
C56
0.047uF
C40
0.047uF
(U17)
C44
0.047uF
C32
0.047uF
(U18)
C63
0.047uF
(U19)
C39
0.047uF
C52
0.047uF
(U24)
C9
0.047uF
(U21)
(U25)
(U26)
C8
0.047uF
C73
0.047uF
(U27)
C69
0.047uF
C70
0.047uF
C66
0.047uF
C54
0.047uF
10uF
+
C68
10uF
10uF
+
C57
10uF
+
C65
+
C55

FIG. 20-2

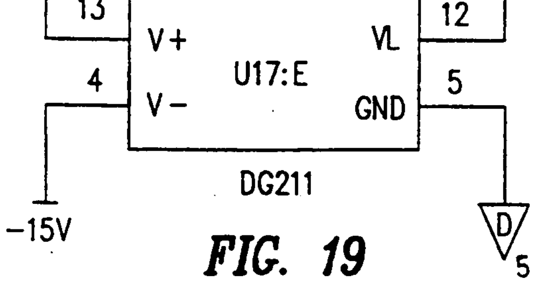

+15V
+5VD
13
V+
U17:E
VL
12
4
V−
GND
5
DG211
−15V
D

FIG. 19

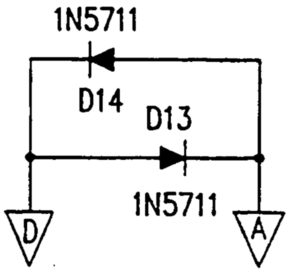

1N5711
D14
D13
1N5711
D
A

FIG. 18

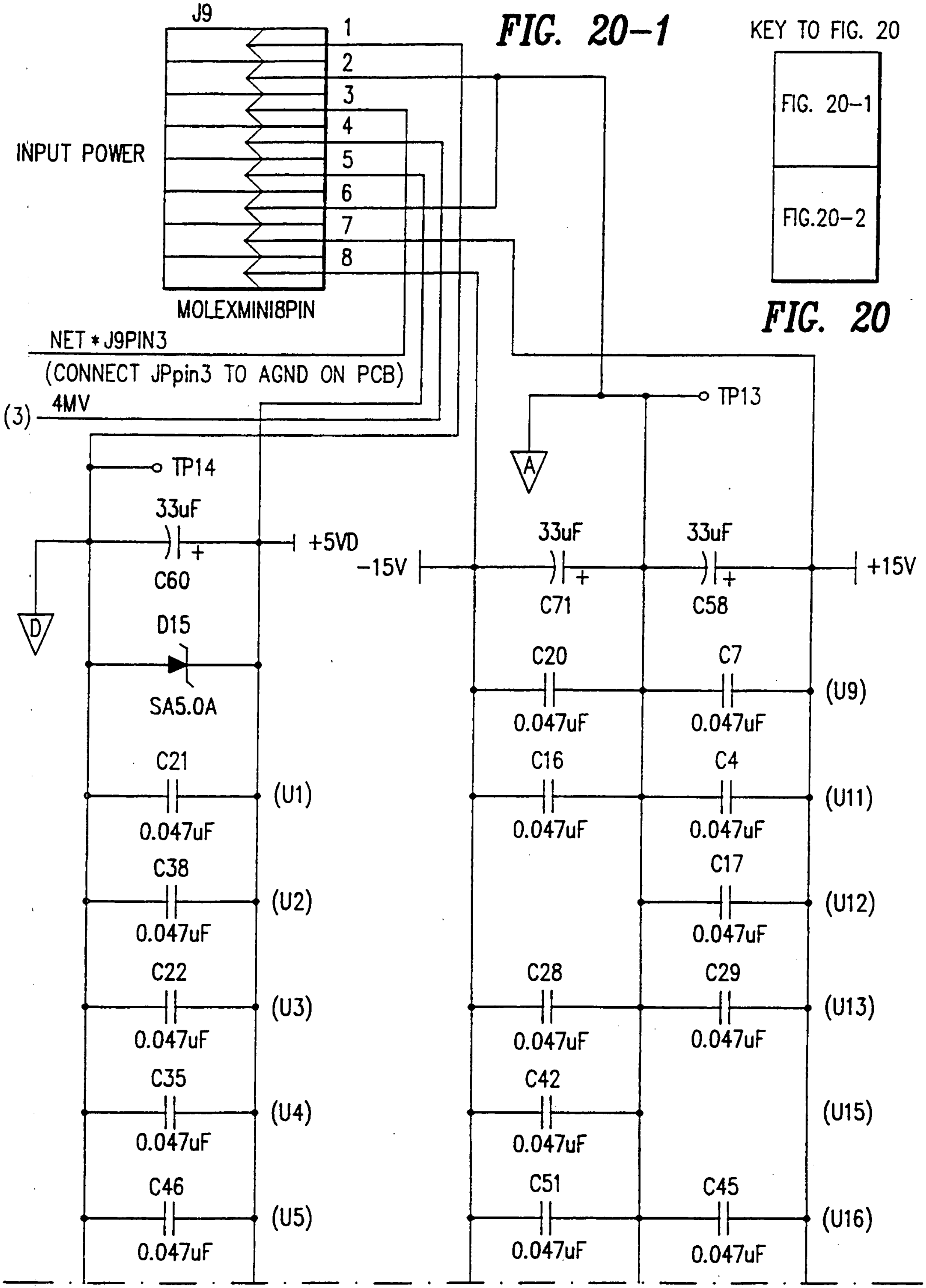
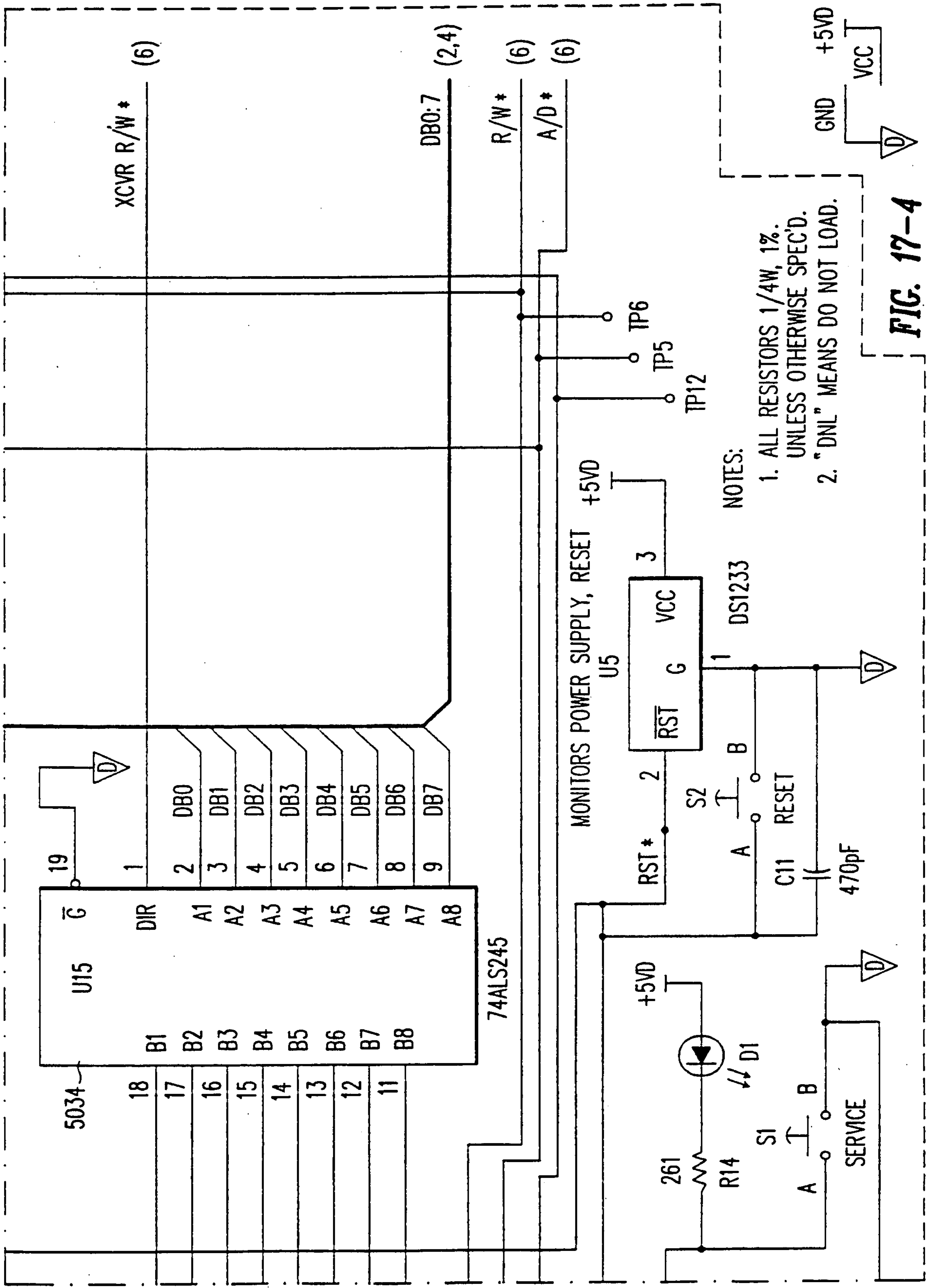


FIG. 20-1

KEY TO FIG. 20

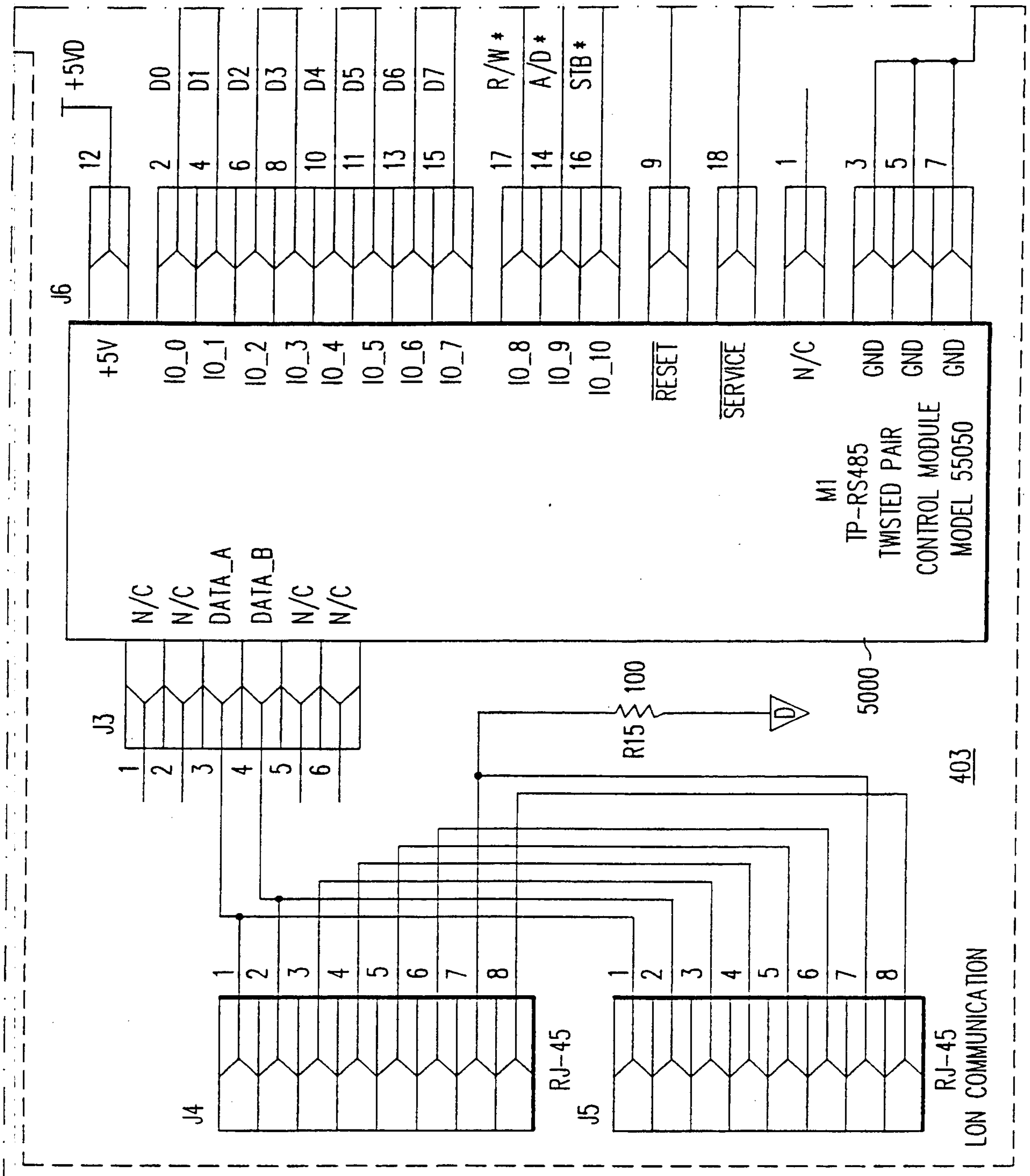
FIG. 20

J9
1
2
3
4
5
6
7
8
INPUT POWER
MOLEXMINI8PIN
NET * J9PIN3
(CONNECT JPpin3 TO AGND ON PCB)
4MV
(3)
TP14
TP13
A
D
33uF
C60
+5VD
D15
SA5.0A
C21
0.047uF
(U1)
C38
0.047uF
(U2)
C22
0.047uF
(U3)
C35
0.047uF
(U4)
C46
0.047uF
(U5)
-15V
33uF
C71
33uF
C58
+15V
C20
0.047uF
C7
0.047uF
(U9)
C16
0.047uF
C4
0.047uF
(U11)
C17
0.047uF
(U12)
C28
0.047uF
C29
0.047uF
(U13)
C42
0.047uF
(U15)
C51
0.047uF
C45
0.047uF
(U16)
FIG. 20-1
FIG.20-2


XCVR R/W*
(6)
DB0:7
(2,4)
R/W*
(6)
A/D*
(6)
GND
+5VD
VCC
TP6
TP5
TP12
+5VD
MONITORS POWER SUPPLY, RESET
U5
DS1233
VCC
G
RST
3
1
2
RST*
S2
RESET
A
B
C11
470pF
19
G
1
DIR
2 3 4 5 6 7 8 9
A1 A2 A3 A4 A5 A6 A7 A8
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7
U15
74ALS245
B1 B2 B3 B4 B5 B6 B7 B8
18 17 16 15 14 13 12 11
5034
+5VD
D1
261
R14
S1
SERVICE
A
B
D

NOTES:
1. ALL RESISTORS 1/4W, 1%.
UNLESS OTHERWISE SPEC'D.
2. "DNL" MEANS DO NOT LOAD.

FIG. 17-4


+5VD
J6
12
2
4
6
8
10
11
13
15
17
14
16
9
18
1
3
5
7
D0
D1
D2
D3
D4
D5
D6
D7
R/W*
A/D*
STB*
+5V
IO_0
IO_1
IO_2
IO_3
IO_4
IO_5
IO_6
IO_7
IO_8
IO_9
IO_10
RESET
SERVICE
N/C
GND
GND
GND
M1
TP-RS485
TWISTED PAIR
CONTROL MODULE
MODEL 55050
N/C
N/C
DATA_A
DATA_B
N/C
N/C
J3
1
2
3
4
5
6
5000
R15 100
D
403
J4
RJ-45
1
2
3
4
5
6
7
8
J5
RJ-45
LON COMMUNICATION
1
2
3
4
5
6
7
8

KEY TO FIG. 17

FIG. 17-1	FIG. 17-2
FIG. 17-3	FIG. 17-4

FIG. 17

FIG. 17-3